

IN THE UNITED STATES PATENT
AND TRADEMARK OFFICE

13

	Reissue Application No.:)	
5	09/512,592)	
	United States Patent No.:)	Group Art Unit: 2177
	5,806,063)	
	Issued: September 8, 1998)	Examiner: Paul Kulik
	Applicant:)	
10	Dickens-Soeder2000, LLC)	Attorney Docket No.:
)	2039-154
	Reexamination Proceeding:)	
	90/005,592)	
	Filed: December 21, 1999)	
15)	
	Reexamination Proceeding:)	
	90/005,628)	
	Filed: February 2, 2000)	
)	
20	Reexamination Proceeding:)	
	90/005,727)	
	Filed: May 16, 2000)	
)	

RECEIVED
JAN 05 2001
REEXAM UNIT

25

30

SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT
SUBMISSION

Honorable Commissioner of Patents and Trademarks
Washington, D.C. 20231

Dear Sir:

5 Pursuant to the DECISION, *SUA SPONTE*, TO MERGE
REEXAMINATION AND REISSUE PROCEEDINGS, dated November
03, 2000 and mailed November 6, 2000 ("the Decision"),
the in the above referenced Reissue Application and
Patent Owner in the above referenced Reexamination
10 Proceedings ("the Applicant"), were merged. Applicant
has submitted an Information disclosure Statement with
respect to the above referenced Reissue Application,
which was incorporated by reference into the
Information disclosure Statement filed by the Applicant
15 in accordance with the requirements of the Decision and
37 C.F.R. §1.565(d). Since the time of the filing of
the Information disclosure Statement in the above
referenced Reissue Application the Applicant has come
into possession of copies of some non-patent documents
20 referenced in the Information disclosure Statement
filed with the Reissue Application as purported
references.

Specifically these copies are of documents listed
on the Internet Web-Site of the Information Technology
25 Association of America. Applicant hereby submits
copies of the following documents from that list and

#13

referenced in the Information disclosure Statement
filed with the Reissue application.

RECEIVED

JAN 0 5 2001

REEXAM UNIT

5 Guy Steel, Common Lisp, The Language, Second Edition,
Digital Equipment Corporation, 1990, p.702.

Guy Steel, Common Lisp, The Language, Digital
Equipment Corporation, 1984.

10 David Moon, et al., Lisp Machine Manual, Sixth
Edition, MIT Artificial Intelligence Laboratory,
1984, p. 776. Note: The Compatibility Note in "Common
Lisp, The Language" refers to an earlier edition of
that language, prior to the introduction of the
rolling "10-decade" window. The fourth edition of the
Lisp Machine Manual, for instance, does not include
15 the feature.

Oracle Corporation, SQL Language Reference Manual,
Version 7.0, Part number 778-70-1292, December 1992,
pp. 3-54 to 3-55.

20 International Standard ISO 8601: 1988 (E), "Data
Elements and Interchange Formats - Information
Exchange - Representation of Dates and Times,"
International Organization for Standardization, 1st
edition, 1988-06-15.

25 International Standard ISO 2014: 1976 (E), "Writing
of Calendar Dates in All-Numeric Form", International
Organization for Standardization, first edition 1976-
04-01.

Jonathan Postel, Internet RFC 753, "Internet Message
Protocol", March 1979.

-
- Richard Bergeon, The Millennium Journal, Vol. II.IV,
July 1995, pp. 1-5. (Two versions)
-
- 5 Gary Browe, "Intelligent Report Maintenance Using
Dialogue Manager", Focus Systems Journal, March 1990,
pp. 70-71.
-
- SAS Institute, Inc., SAS Language: Reference, Version
6, First Edition, Cary, NC; SAS Institute Inc., 1990,
p. 63, 68, 85, 182, 536-539, 649-655, 670, 682-685,
690-695, 698-699, 704-706, 746, 790-791.
-
- 10 Stan Milam, "Extended Data Library for C", C-C++
Users Journal, Vol. 12, No. 10, October 1994, p. 67-
80.
-
- 15 "Here's One Alternative Date Method", Tick, Tick,
Tick, Vol. 1, No. 1, 1993, p1 and "One Man's
Opinions," p. 5.
-
- Anon, IBM, "Date Adjustment At The Turn Of The
Century", TDB, May 1986, N265 05-6 [Freeman].
-
- DeJager, "Doomsday 2000", Computerworld, Sept. 6,
1993. (Two versions)
-
- 20 Furman, "Party When It's 1999", SOFTWARE MAGAZINE,
April 1995, p.6.
-
- Gilllin, "A Y2K Pioneer Seeks (And Deserves)
Recognition", 1984, Schoen. (Two versions)
-
- 25 Glass, Robert, "The Next Date Crisis and the Ones
After That", Comm of the ACM, v 40, #1, Jan 1997.
-
- Greer, "Method of Sorting Dates and Time Allowing for
Wrapping", IBM TDB, v 37, #8, August 1994, p. 381-2.
-

Hart et al., "A Scaleable, Automated Process for Year 2000 System Correction", PROC 18th International Conference on Software Engineering, 25-30 Mar. 1996, pp 475-484, Mar 30, 1996.

5 Hayes, "Waiting for 01-01-00", AMERICAN SCIENTIST, v 83, #1, Jan. 1995.

Matthews, "Excel 4 for Windows", 1992, p. 347.

Meyer, "Julian and Gregorian Calendars", Dr. Dobbs Journal, March 1993.

10 Neuhaus, "Databases", PC Magazine, v 9, #16, p. 471, Sept. 25, 1990.

Pieptea, D.R., "What Will The Change Of The Millennium Do To Our Data Processing", Management Information Systems Quarterly, v 10, #2, pp 103-4, June 1, 1986.

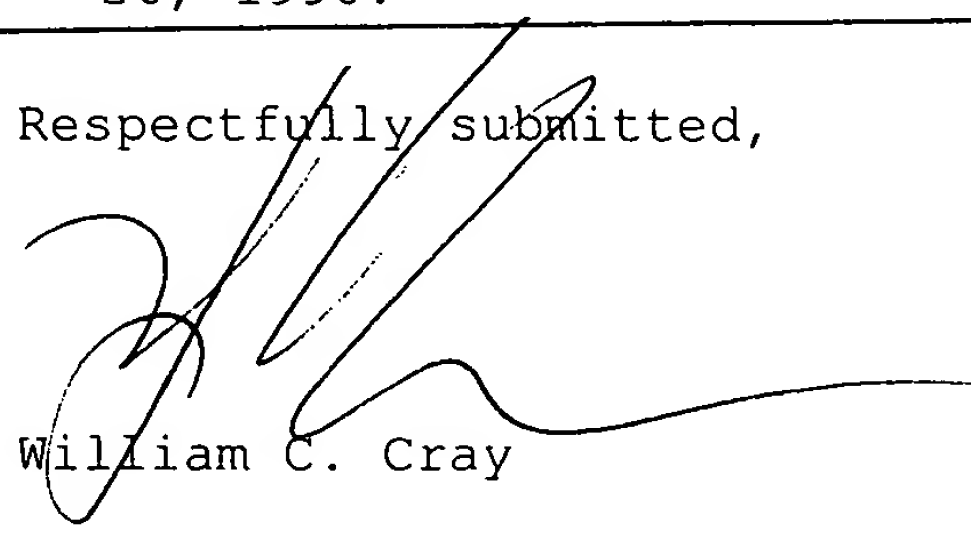
Roberts, "DataServer 2000: Computer Software Prepares Legacy Systems for Year 2000", Enterprise Systems Journal, Nov. 1994, p. 10.

20 Schoen, "The Charmar Correction", copyright registration TX 144-098, Dec. 1983.

Smith, H. J., "What's Ahead For 2000 AD?", APL Quote Quad, v 20, #4, p. 364, July 1990.

Xenakis, "Preparing For 2000", INFORMATION WEEK, Feb. 26, 1990.

25 Respectfully submitted,


William C. Cray

#13

Reg.No. 27,627

COMMON LISP

THE LANGUAGE

SECOND EDITION

GUY L. STEELE JR.

with contributions by

SCOTT E. FAHLMAN

RICHARD P. GABRIEL

DAVID A. MOON

DANIEL L. WEINREB

and with contributions to the second edition by

DANIEL G. BOBROW

LINDA G. DEMICHIEL

RICHARD P. GABRIEL

SONYA E. KEENE

GREGOR KICZALES

DAVID A. MOON

CRISPIN PERDUE

KENT M. PITMAN

RICHARD C. WATERS

JON L. WHITE

ENGR

001.6424

L695

1990

C. 2

Copyright 1990 by Digital Equipment Corporation.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

9 8 7 6 5 4 3 2 1

Printed in the United States of America.

ADA used to be a registered trademark of the U.S. Government Ada Joint Program Office, but is no longer a trademark. Chevrolet is a registered trademark of General Motors. DEC, PDP, VAX, and VMS are trademarks of Digital Equipment Corporation. Hostess Twinkies is a registered trademark of Continental Baking Company. IBM is a registered trademark of International Business Machines Corporation. MacIntosh is a registered trademark of Apple Computer, Inc. Multics is a registered trademark of Honeywell Inc. PostScript is a registered trademark of Adobe Systems Incorporated. Rolo is a registered trademark of Hershey Foods Corporation. SPAM is a registered trademark of Geo. A. Hormel & Co. Tang is a brand name of General Foods Corporation. Teenage Mutant Ninja Turtles is a registered trademark, and Donatello, Krang, Leonardo, Michaelangelo, Raphael, and Shredder are trademarks of Mirage Studios, USA. Teflon is a registered trademark of Du Pont. T_EX is a trademark of the American Mathematical Society. UNIX is a trademark of Bell Laboratories. ZETALISP is a registered trademark of Symbolics, Inc. All other trademarks are the property of their respective owners.

Design: David Ford

Copyediting: Alice Cheyer and Kate Schmit

Index: Marilyn Rowland

Composition: Guy L. Steele Jr. (See Colophon.)

Camera-ready copy: Advanced Computer Graphics

Production: Editorial Inc.

Printer: Hamilton Printing Company

Library of Congress Cataloging-in-Publication Data

Steele, Guy.

COMMON LISP.

Includes bibliographical references (p.)

1. COMMON LISP (Computer program language) I. Title.

QA76.73.L23S73 1990 005.13'3 89-26016

Casebound order number EY-C194E-DP ISBN 1-55558-042-4

Pre:

Act

Act

1.

1.1

1.2

1.2

1.2

1.2

1.2

1.2

1.2

1.2

2.

2.1

2.1

2.1

2.1

2.1

2.2

2.2

2.2

2.2

2.2

2.2

2.2

2.2

2.2

284472

25.4.1. Time Functions

Time is represented in three different ways in Common Lisp: Decoded Time, Universal Time, and Internal Time. The first two representations are used primarily to represent calendar time and are precise only to one second. Internal Time is used primarily to represent measurements of computer time (such as run time) and is precise to some implementation-dependent fraction of a second, as specified by `internal-time-units-per-second`. Decoded Time format is used only for absolute time indications. Universal Time and Internal Time formats are used for both absolute and relative times.

Decoded Time format represents calendar time as a number of components:

- *Second*: an integer between 0 and 59, inclusive.
- *Minute*: an integer between 0 and 59, inclusive.
- *Hour*: an integer between 0 and 23, inclusive.
- *Date*: an integer between 1 and 31, inclusive (the upper limit actually depends on the month and year, of course).
- *Month*: an integer between 1 and 12, inclusive; 1 means January, 12 means December.
- *Year*: an integer indicating the year A.D. However, if this integer is between 0 and 99, the "obvious" year is used; more precisely, that year is assumed that is equal to the integer modulo 100 and within fifty years of the current year (inclusive backwards and exclusive forwards). Thus, in the year 1978, year 28 is 1928 but year 27 is 2027. (Functions that return time in this format always return a full year number.)

Compatibility note: This is incompatible with the Lisp Machine Lisp definition in two ways. First, in Lisp Machine Lisp a year between 0 and 99 always has 1900 added to it. Second, in Lisp Machine Lisp time functions return the abbreviated year number between 0 and 99 rather than the full year number. The incompatibility is prompted by the imminent arrival of the twenty-first century. Note that `(mod year 100)` always reliably converts a year number to the abbreviated form, while the inverse conversion can be very difficult.

- *Day-of-week*: an integer between 0 and 6, inclusive; 0 means Monday, 1 means Tuesday, and so on; 6 means Sunday.
- *Daylight-saving-time-p*: a flag that, if not `nil`, indicates that daylight saving time is in effect.

second, minute, hour, date, month, year, day-of-week, daylight-saving-time-p, and time-zone.

Compatibility note: In Lisp Machine Lisp *time-zone* is not currently returned. Consider, however, the use of Common Lisp in some mobile vehicle. It is entirely plausible that the time zone might change from time to time.

`get-universal-time`

[Function]

The current time of day is returned as a single integer in Universal Time format.

`decode-universal-time` *universal-time* &optional *time-zone* [Function]

The time specified by *universal-time* in Universal Time format is converted to Decoded Time format. Nine values are returned: *second, minute, hour, date, month, year, day-of-week, daylight-saving-time-p, and time-zone.*

Compatibility note: In Lisp Machine Lisp *time-zone* is not currently returned. Consider, however, the use of Common Lisp in some mobile vehicle. It is entirely plausible that the time zone might change from time to time.

The *time-zone* argument defaults to the current time zone.

X3J13 voted in January 1989 (47) to specify that `decode-universal-time`, like `encode-universal-time`, ignores daylight saving time information if a *time-zone* is explicitly specified; in this case the returned *daylight-saving-time-p* value will necessarily be `nil` even if daylight saving time happens to be in effect in that time zone at the specified time.

`encode-universal-time` *second minute hour date month year* [Function]
&optional *time-zone*

The time specified by the given components of Decoded Time format is encoded into Universal Time format and returned. If you do not specify *time-zone*, it defaults to the current time zone adjusted for daylight saving time. If you provide *time-zone* explicitly, no adjustment for daylight saving time is performed.

`internal-time-units-per-second`

[Constant]

This value is an integer, the implementation-dependent number of internal time units in a second. (The internal time unit must be chosen so that one second is an integral multiple of it.)

Rational
so that g
overhead
value fro
off the s
for some
much sn
get-int

get-in

The cur
precise
time, ru
between
the two
executir

get-in

The cur
is relati
calls to
measure

sleep :

(sleep
seconds
non-neg

25.4.2.

For any
produce

Rational
that a Co

- *Time-zone*: an integer specified as the number of hours west of GMT (Greenwich Mean Time). For example, in Massachusetts the time zone is 5, and in California it is 8. Any adjustment for daylight saving time is separate from this.

X3J13 voted in March 1989 (178) to specify that the time zone part of Decoded Time need not be an integer, but may be any rational number (either an integer or a ratio) in the range -24 to 24 (inclusive on both ends) that is an integral multiple of 1/3600.

Rationale: For all possible time designations to be accommodated, it is necessary to allow the time zone to be non-integral, for some places in the world have time standards offset from Greenwich Mean Time by a non-integral number of hours.

There appears to be no user demand for floating-point time zones. Since such zones would introduce inexact arithmetic, X3J13 did not consider adding them at this time.

This specification does require time zones to be represented as integral multiples of 1 second (rather than 1 hour). This prevents problems that could otherwise occur in converting Decoded Time to Universal Time.

Universal Time represents time as a single non-negative integer. For relative time purposes, this is a number of seconds. For absolute time, this is the number of seconds since midnight, January 1, 1900 GMT. Thus the time 1 is 00:00:01 (that is, 12:00:01 A.M.) on January 1, 1900 GMT. Similarly, the time 2398291201 corresponds to time 00:00:01 on January 1, 1976 GMT. Recall that the year 1900 was *not* a leap year; for the purposes of Common Lisp, a year is a leap year if and only if its number is divisible by 4, except that years divisible by 100 are *not* leap years, except that years divisible by 400 *are* leap years. Therefore the year 2000 will be a leap year. (Note that the "leap seconds" that are sporadically inserted by the world's official timekeepers as an additional correction are ignored; Common Lisp assumes that every day is exactly 86400 seconds long.) Universal Time format is used as a standard time representation within the ARPANET; see reference [22]. Because the Common Lisp Universal Time representation uses only non-negative integers, times before the base time of midnight, January 1, 1900 GMT cannot be processed by Common Lisp.

Internal Time also represents time as a single integer, but in terms of an implementation-dependent unit. Relative time is measured as a number of these units. Absolute time is relative to an arbitrary time base, typically the time at which the system began running.

get-decoded-time

[Function]

The current time is returned in Decoded Time format. Nine values are returned:

Rationale: The reason for allowing the internal time units to be implementation-dependent is so that `get-internal-run-time` and `get-internal-real-time` can execute with minimum overhead. The idea is that it should be very likely that a fixnum will suffice as the returned value from these functions. This probability can be tuned to the implementation by trading off the speed of the machine against the word size. Any particular unit will be inappropriate for some implementations: a microsecond is too long for a very fast machine, while a much smaller unit would force many implementations to return bignums for most calls to `get-internal-time`, rendering that function less useful for accurate timing measurements.

`get-internal-run-time`

[Function]

The current run time is returned as a single integer in Internal Time format. The precise meaning of this quantity is implementation-dependent; it may measure real time, run time, CPU cycles, or some other quantity. The intent is that the difference between the values of two calls to this function be the amount of time between the two calls during which computational effort was expended on behalf of the executing program.

`get-internal-real-time`

[Function]

The current time is returned as a single integer in Internal Time format. This time is relative to an arbitrary time base, but the difference between the values of two calls to this function will be the amount of elapsed real time between the two calls, measured in the units defined by `internal-time-units-per-second`.

`sleep seconds`

[Function]

(`sleep n`) causes execution to cease and become dormant for approximately *n* seconds of real time, whereupon execution is resumed. The argument may be any non-negative non-complex number. `sleep` returns `nil`.

25.4.2. Other Environment Inquiries

For any of the following functions, if no appropriate and relevant result can be produced, `nil` is returned instead of a string.

Rationale: These inquiry facilities are functions rather than variables against the possibility that a Common Lisp process might migrate from machine to machine. This need not happen

#13

③

COMMON LISP

THE LANGUAGE

GUY L. STEELE JR.

Carnegie-Mellon University
Tartan Laboratories Incorporated

with contributions by

SCOTT E. FAHLMAN

Carnegie-Mellon University

RICHARD P. GABRIEL

Stanford University
Lawrence Livermore National Laboratory

DAVID A. MOON

Symbolics, Incorporated

DANIEL L. WEINREB

Symbolics, Incorporated

Math

001.6424

L69s

c.2

digital

Digital Press

Copyright © 1984 by Digital Equipment Corporation.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

9 8 7 6 5 4

Printed in the United States of America.

Designed by David Ford. Automatically typeset from magnetic tape by Waldman Graphics, Pennsauken, New Jersey. Printed and bound by Halliday Lithographers, Hanover, Massachusetts.

Order number EY-00031-DP

"ADA" is a registered trademark of the U.S. Government—Ada Joint Program Office. "DEC," "PDP," "VAX," and "VMS" are trademarks of Digital Equipment Corporation. "IBM" is a registered trademark of International Business Machines Corporation. "Multics" is a registered trademark of Honeywell Inc. "Tang" is a brand name of General Foods Corporation. "UNIX" is a trademark of Bell Laboratories.

Library of Congress Cataloging in Publication Data

Steele, Guy.

Common LISP: The Language.

Includes bibliographical references and index.

I. LISP (Computer program language) I. Title.

II. Title: Common LISP: The Language.

QA76.73.L23S73 1984 001.64'24 84-7681

ISBN 0-932376-41-X

17704

Contents

1. Introduction 1

- 1.1. Purpose 1
- 1.2. Notational Conventions 4
 - 1.2.1. Decimal Numbers 4
 - 1.2.2. Nil, False, and the Empty List 4
 - 1.2.3. Evaluation, Expansion, and Equivalence 4
 - 1.2.4. Errors 5
 - 1.2.5. Descriptions of Functions and Other Entities 6
 - 1.2.6. The Lisp Reader 8
 - 1.2.7. Overview of Syntax 9

2. Data Types 11

- 2.1. Numbers 13
 - 2.1.1. Integers 13
 - 2.1.2. Ratios 15
 - 2.1.3. Floating-point Numbers 16
 - 2.1.4. Complex Numbers 19
- 2.2. Characters 20
 - 2.2.1. Standard Characters 20
 - 2.2.2. Line Divisions 21
 - 2.2.3. Non-standard Characters 23
 - 2.2.4. Character Attributes 23
 - 2.2.5. String Characters 23
- 2.3. Symbols 23
- 2.4. Lists and Conses 26

- 2.5. Arrays 28
 - 2.5.1. Vectors 29
 - 2.5.2. Strings 30
 - 2.5.3. Bit-Vectors 30
- 2.6. Hash Tables 31
- 2.7. Readtables 31
- 2.8. Packages 31
- 2.9. Pathnames 31
- 2.10. Streams 31
- 2.11. Random-States 31
- 2.12. Structures 32
- 2.13. Functions 32
- 2.14. Unreadable Data Objects 32
- 2.15. Overlap, Inclusion, and Disjointness of Types 33

3. Scope and Extent 36

4. Type Specifiers 42

- 4.1. Type Specifier Symbols 42
- 4.2. Type Specifier Lists 42
- 4.3. Predicating Type Specifiers 43
- 4.4. Type Specifiers that Combine 44
- 4.5. Type Specifiers that Specialize 45
- 4.6. Type Specifiers that Abbreviate 48

4.7.	Defining New Type Specifiers	50	7.5.	Establishing New Variable Bindings	110
4.8.	Type Conversion Function	51	7.6.	Conditionals	114
4.9.	Determining the Type of an Object	52	7.7.	Blocks and Exits	119
✓ 5.	Program Structure	54	7.8.	Iteration	121
5.1.	Forms	54	7.8.1.	Indefinite Iteration	121
5.1.1.	Self-Evaluating Forms	55	7.8.2.	General Iteration	121
5.1.2.	Variables	55	7.8.3.	Simple Iteration Constructs	126
5.1.3.	Special Forms	56	7.8.4.	Mapping	128
5.1.4.	Macros	57	7.8.5.	The "Program Feature"	130
5.1.5.	Function Calls	58	7.9.	Multiple Values	133
5.2.	Functions	59	7.9.1.	Constructs for Handling Multiple Values	133
5.2.1.	Named Functions	59	7.9.2.	Rules Governing the Passing of Multiple Values	137
5.2.2.	Lambda-Expressions	59	7.10.	Dynamic Non-local Exits	139
5.3.	Top-Level Forms	66	✓ 8.	Macros	
5.3.1.	Defining Named Functions	67	8.1.	Macro Definition	144
5.3.2.	Declaring Global Variables and Named Constants	68	8.2.	Macro Expansion	151
5.3.3.	Control of Time of Evaluation	69	9.	Declarations	153
✓ 6.	Predicates	71	9.1.	Declaration Syntax	153
6.1.	Logical Values	72	9.2.	Declaration Specifiers	157
6.2.	Data Type Predicates	72	9.3.	Type Declaration for Forms	161
6.2.1.	General Type Predicates	72	✓ 10.	Symbols	163
6.2.2.	Specific Data Type Predicates	73	10.1.	The Property List	163
6.3.	Equality Predicates	77	10.2.	The Print Name	167
6.4.	Logical Operators	82	10.3.	Creating Symbols	168
✓ 7.	Control Structure	85	11.	Packages	171
7.1.	Constants and Variables	86	11.1.	Consistency Rules	172
7.1.1.	Reference	86	11.2.	Package Names	173
7.1.2.	Assignment	91	11.3.	Translating Strings to Symbols	174
7.2.	Generalized Variables	93	11.4.	Exporting and Importing Symbols	176
7.3.	Function Invocation	107			
7.4.	Simple Sequencing	108			

- 4.7. Defining New Type Specifiers 50
- 4.8. Type Conversion Function 51
- 4.9. Determining the Type of an Object 52

✓ 5. Program Structure 54

- 5.1. Forms 54
 - 5.1.1. Self-Evaluating Forms 55
 - 5.1.2. Variables 55
 - 5.1.3. Special Forms 56
 - 5.1.4. Macros 57
 - 5.1.5. Function Calls 58
- 5.2. Functions 59
 - 5.2.1. Named Functions 59
 - 5.2.2. Lambda-Expressions 59
- 5.3. Top-Level Forms 66
 - 5.3.1. Defining Named Functions 67
 - 5.3.2. Declaring Global Variables and Named Constants 68
 - 5.3.3. Control of Time of Evaluation 69

✓ 6. Predicates 71

- 6.1. Logical Values 72
- 6.2. Data Type Predicates 72
 - 6.2.1. General Type Predicates 72
 - 6.2.2. Specific Data Type Predicates 73
- 6.3. Equality Predicates 77
- 6.4. Logical Operators 82

✓ 7. Control Structure 85

- 7.1. Constants and Variables 86
 - 7.1.1. Reference 86
 - 7.1.2. Assignment 91
- 7.2. Generalized Variables 93
- 7.3. Function Invocation 107
- 7.4. Simple Sequencing 108

- 7.5. Establishing New Variable Bindings 110
- 7.6. Conditionals 114
- 7.7. Blocks and Exits 119
- 7.8. Iteration 121
 - 7.8.1. Indefinite Iteration 121
 - 7.8.2. General Iteration 121
 - 7.8.3. Simple Iteration Constructs 126
 - 7.8.4. Mapping 128
 - 7.8.5. The "Program Feature" 130
- 7.9. Multiple Values 133
 - 7.9.1. Constructs for Handling Multiple Values 133
 - 7.9.2. Rules Governing the Passing of Multiple Values 137
- 7.10. Dynamic Non-local Exits 139

✓ 8. Macros

- 8.1. Macro Definition 144
- 8.2. Macro Expansion 151

9. Declarations 153

- 9.1. Declaration Syntax 153
- 9.2. Declaration Specifiers 157
- 9.3. Type Declaration for Forms 161

✓ 10. Symbols 163

- 10.1. The Property List 163
- 10.2. The Print Name 167
- 10.3. Creating Symbols 168

11. Packages 171

- 11.1. Consistency Rules 172
- 11.2. Package Names 173
- 11.3. Translating Strings to Symbols 174
- 11.4. Exporting and Importing Symbols 176

- 11.5. Name Conflicts 178
- 11.6. Built-in Packages 181
- 11.7. Package System Functions and Variables 182
- 11.8. Modules 188
- 11.9. An Example 189

12. Numbers 193

- 12.1. Precision, Contagion, and Coercion 193
- 12.2. Predicates on Numbers 195
- 12.3. Comparisons on Numbers 196
- 12.4. Arithmetic Operations 199
- 12.5. Irrational and Transcendental Functions 203
 - 12.5.1. Exponential and Logarithmic Functions 203
 - 12.5.2. Trigonometric and Related Functions 205
 - 12.5.3. Branch Cuts, Principal Values, and Boundary Conditions in the Complex Plane 210
- 12.6. Type Conversions and Component Extractions on Numbers 214
- 12.7. Logical Operations on Numbers 220
- 12.8. Byte Manipulation Functions 225
- 12.9. Random Numbers 228
- 12.10. Implementation Parameters 231

13. Characters 233

- 13.1. Character Attributes 233
- 13.2. Predicates on Characters 234
- 13.3. Character Construction and Selection 239
- 13.4. Character Conversions 241
- 13.5. Character Control-Bit Functions 243

14. Sequences 245

- 14.1. Simple Sequence Functions 247
- 14.2. Concatenating, Mapping, and Reducing Sequences 249
- 14.3. Modifying Sequences 252
- 14.4. Searching Sequences for Items 256
- 14.5. Sorting and Merging 258

15. Lists 262

- 15.1. Conses 262
- 15.2. Lists 264
- 15.3. Alteration of List Structure 272
- 15.4. Substitution of Expressions 273
- 15.5. Using Lists as Sets 275
- 15.6. Association Lists 279

16. Hash Tables 282

- 16.1. Hash Table Functions 283
- 16.2. Primitive Hash Function 285

17. Arrays 286

- 17.1. Array Creation 286
- 17.2. Array Access 290
- 17.3. Array Information 291
- 17.4. Functions on Arrays of Bits 293
- 17.5. Fill Pointers 295
- 17.6. Changing the Dimensions of an Array 297

18. Strings 299

- 18.1. String Access 299
- 18.2. String Comparison 300
- 18.3. String Construction and Manipulation 302

✓ **19. Structures 305**

- 19.1. Introduction to Structures 305
- 19.2. How to Use Defstruct 307
- 19.3. Using the Automatically Defined Constructor Function 309
- 19.4. Defstruct Slot-Options 310
- 19.5. Defstruct Options 311
- 19.6. By-position Constructor Functions 315
- 19.7. Structures of Explicitly Specified Representational Type 316
 - 19.7.1. Unnamed Structures 317
 - 19.7.2. Named Structures 318
 - 19.7.3. Other Aspects of Explicitly Specified Structures 319

✓ **20. The Evaluator 321**

- 20.1. Run-Time Evaluation of Forms 321
- 20.2. The Top-Level Loop 324

✓ **21. Streams 327**

- 21.1. Standard Streams 327
- 21.2. Creating New Streams 329
- 21.3. Operations on Streams 332

✓ **22. Input/Output 333**

- 22.1. Printed Representation of LISP Objects 333
 - 22.1.1. What the Read Function Accepts 334
 - 22.1.2. Parsing of Numbers and Symbols 339
 - 22.1.3. Macro Characters 346
 - 22.1.4. Standard Dispatching Macro Character Syntax 351
 - 22.1.5. The Readtable 360
 - 22.1.6. What the Print Function Produces 365
- 22.2. Input Functions 374

- 22.2.1. Input from Character Streams 374
- 22.2.2. Input from Binary Streams 382
- 22.3. Output Functions 382
 - 22.3.1. Output to Character Streams 382
 - 22.3.2. Output to Binary Streams 385
 - 22.3.3. Formatted Output to Character Streams 385
- 22.4. Querying the User 407

23. File System Interface 409

- 23.1. File Names 409
 - 23.1.1. Pathnames 410
 - 23.1.2. Pathname Functions 413
- 23.2. Opening and Closing Files 418
- 23.3. Renaming, Deleting, and Other File Operations 423
- 23.4. Loading Files 426
- 23.5. Accessing Directories 427

24. Errors 428

- 24.1. General Error-Signalling Functions 429
- 24.2. Specialized Error-Signalling Forms and Macros 433
- 24.3. Special Forms for Exhaustive Case Analysis 435

✓ **25. Miscellaneous Features 438**

- 25.1. The Compiler 438
- 25.2. Documentation 439
- 25.3. Debugging Tools 440
- 25.4. Environment Inquiries 443
 - 25.4.1. Time Functions 443
 - 25.4.2. Other Environment Inquiries 447
- 25.5. Identity Function 448

References 449
Index 451

`dribble` *&optional pathname* [Function]

(`dribble` *pathname*) rebinds `*standard-input*` and `*standard-output*`, and/or takes other appropriate action, so as to send a record of the input/output interaction to a file named by *pathname*. The primary purpose of this is to create a readable record of an interactive session.

(`dribble`) terminates the recording of input and output and closes the dribble file.

`apropos` *string* *&optional package* [Function]

`apropos-list` *string* *&optional package* [Function]

(`apropos` *string*) tries to find all available symbols whose print names contain *string* as a substring. (A symbol may be supplied for the *string*, in which case the print name of the symbol is used.) Whenever `apropos` finds a symbol, it prints out the symbol's name; in addition, information about the function definition and dynamic value of the symbol, if any, is printed. If *package* is specified and not `nil`, then only symbols available in that package are examined; otherwise "all" packages are searched, as if by `do-all-symbols`. Because a symbol may be available by way of more than one inheritance path, `apropos` may print information about the same symbol more than once. The information is printed to the stream that is the value of `*standard-output*`. `apropos` returns no values (that is, it returns what the expression (`values`) returns: zero values).

`apropos-list` performs the same search that `apropos` does, but prints nothing. It returns a list of the symbols whose print names contain *string* as a substring.

25.4. Environment Inquiries

Environment inquiry functions provide information about the environment in which a COMMON LISP program is being executed. They are described here in two categories: first, those dealing with determination and measurement of time, and second, all the others, most of which deal with identification of the computer hardware and software.

25.4.1. Time Functions

Time is represented in three different ways in COMMON LISP: Decoded Time, Universal Time, and Internal Time. The first two representations are used primarily to represent calendar time, and are precise only to one second. Internal Time is

used primarily to represent measurements of computer time (such as run time) and is precise to some implementation-dependent fraction of a second, as specified by `internal-time-units-per-second`. Decoded Time format is used only for absolute time indications. Universal Time and Internal Time formats are used for both absolute and relative times.

Decoded Time format represents calendar time as a number of components:

- *Second*: an integer between 0 and 59, inclusive.
- *Minute*: an integer between 0 and 59, inclusive.
- *Hour*: an integer between 0 and 23, inclusive.
- *Date*: an integer between 1 and 31, inclusive (the upper limit actually depends on the month and year, of course).
- *Month*: an integer between 1 and 12, inclusive; 1 means January, 12 means December.
- *Year*: an integer indicating the year A.D. However, if this integer is between 0 and 99, the "obvious" year is used; more precisely, that year is assumed that is equal to the integer modulo 100 and within fifty years of the current year (inclusive backwards and exclusive forwards). Thus, in the year 1978, year 28 is 1928 but year 27 is 2027. (Functions that return time in this format always return a full year number.)

Compatibility note: This is incompatible with the ZETALISP definition in two ways. First, in ZETALISP a year between 0 and 99 always has 1900 added to it. Second, in ZETALISP time functions return the abbreviated year number between 0 and 99 rather than the full year number. The incompatibility is prompted by the imminent arrival of the twenty-first century. Note that `(mod year 100)` always reliably converts a year number to the abbreviated form, while the inverse conversion can be very difficult.

- *Day-of-week*: an integer between 0 and 6, inclusive; 0 means Monday, 1 means Tuesday, and so on; 6 means Sunday.
- *Daylight-saving-time-p*: a flag that, if not `nil`, indicates that daylight saving time is in effect.
- *Time-zone*: an integer specified as the number of hours west of GMT (Greenwich Mean Time). For example, in Massachusetts the time zone is 5, and in California it is 8. Any adjustment for daylight saving time is separate from this.

Universal Time represents time as a single non-negative integer. For relative time purposes, this is a number of seconds. For absolute time, this is the number of seconds since midnight, January 1, 1900 GMT. Thus the time 1 is 00:00:01

(that is, 12:00:01 A.M.) on January 1, 1900 GMT. Similarly, the time 2398291201 corresponds to time 00:00:01 on January 1, 1976 GMT. Recall that the year 1900 was *not* a leap year; for the purposes of COMMON LISP, a year is a leap year if and only if its number is divisible by 4, except that years divisible by 100 are *not* leap years, except that years divisible by 400 *are* leap years. Therefore the year 2000 will be a leap year. (Note that the "leap seconds" that are sporadically inserted by the world's official timekeepers as an additional correction are ignored; COMMON LISP assumes that every day is exactly 86400 seconds long.) Universal Time format is used as a standard time representation within the ARPANET; see reference [8]. Because the COMMON LISP Universal Time representation uses only non-negative integers, times before the base time of midnight, January 1, 1900 GMT cannot be processed by COMMON LISP.

Internal Time also represents time as a single integer, in terms of an implementation-dependent unit. Relative time is measured as a number of these units. Absolute time is relative to an arbitrary time base, typically the time at which the system began running.

`get-decoded-time`

[Function]

The current time is returned in Decoded Time format. Nine values are returned: *second*, *minute*, *hour*, *date*, *month*, *year*, *day-of-week*, *daylight-saving-time-p*, and *time-zone*.

Compatibility note: In ZETALISP the *time-zone* is not currently returned. Consider, however, the use of COMMON LISP in some mobile vehicle. It is entirely plausible that the time zone might change from time to time.

`get-universal-time`

[Function]

The current time of day is returned as a single integer in Universal Time format.

`decode-universal-time` *universal-time* *optional time-zone* [Function]

The time specified by *universal-time* in Universal Time format is converted to Decoded Time format. Nine values are returned: *second*, *minute*, *hour*, *date*, *month*, *year*, *day-of-week*, *daylight-saving-time-p*, and *time-zone*.

Compatibility note: In ZETALISP the *time-zone* is not currently returned. Consider, however, the use of COMMON LISP in some mobile vehicle. It is entirely plausible that the time-zone might change from time to time.

The *time-zone* argument defaults to the current time zone.

```

encode-universal-time second minute hour date month year [Function]
                        &optional time-zone

```

The time specified by the given components of Decoded Time format is encoded into Universal Time format and returned. If you don't specify *time-zone*, it defaults to the current time zone adjusted for daylight saving time. If you provide *time-zone* explicitly, no adjustment for daylight saving time is performed.

internal-time-units-per-second [Constant]

This value is an integer, the implementation-dependent number of internal time units in a second. (The internal time unit must be chosen so that one second is an integral multiple of it.)

Rationale: The reason for allowing the internal time units to be implementation-dependent is so that `get-internal-run-time` and `get-internal-real-time` can execute with minimum overhead. The idea is that it should be very likely that a fixnum will suffice as the returned value from these functions. This probability can be tuned to the implementation by trading off the speed of the machine against the word size. Any particular unit will be inappropriate for some implementations: a microsecond is too long for a very fast machine, while a much smaller unit would force many implementations to return bignums for most calls to `get-internal-time`, rendering that function less useful for accurate timing measurements.

get-internal-run-time [Function]

The current run time is returned as a single integer in Internal Time format. The precise meaning of this quantity is implementation-dependent; it may measure real time, run time, CPU cycles, or some other quantity. The intent is that the difference between the values of two calls to this function be the amount of time between the two calls during which computational effort was expended on behalf of the executing program.

get-internal-real-time [Function]

The current time is returned as a single integer in Internal Time format. This time is relative to an arbitrary time base, but the difference between the values of two calls to this function will be the amount of elapsed real time between the two calls, measured in the units defined by `internal-time-units-per-second`.

`sleep seconds`

[Function]

(`sleep n`) causes execution to cease and become dormant for approximately *n* seconds of real time, whereupon execution is resumed. The argument may be any non-negative non-complex number. `sleep` returns `nil`.

25.4.2. Other Environment Inquiries

For any of the following functions, if no appropriate and relevant result can be produced, `nil` is returned instead of a string.

Rationale: These inquiry facilities are functions rather than variables against the possibility that a COMMON LISP process might migrate from machine to machine. This need not happen in a distributed environment; consider, for example, dumping a core image file containing a compiler and then shipping it to another site.

`lisp-implementation-type`

[Function]

A string is returned that identifies the generic name of the particular COMMON LISP implementation. Examples: "Spice LISP", "Zetalisp".

`lisp-implementation-version`

[Function]

A string is returned that identifies the version of the particular COMMON LISP implementation; this information should be of use to maintainers of the implementation. Examples: "1192", "53.7 with complex numbers", "1746.9A, NEWIO 53, ETHER 5.3".

`machine-type`

[Function]

A string is returned that identifies the generic name of the computer hardware on which COMMON LISP is running. Examples: "DEC PDP-10", "DEC VAX-11/780".

`machine-version`

[Function]

A string is returned that identifies the version of the computer hardware on which COMMON LISP is running. Example: "KL10, microcode 9".

`machine-instance`

[Function]

A string is returned that identifies the particular instance of the computer hardware

4

13

Lisp Machine Manual

Sixth Edition, System Version 99

June 1984

Richard Stallman

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research Contract number N00014-80-C-0505.

34. Dates and Times

The time package contains a set of functions for manipulating dates and times: finding the current time, reading and printing dates and times, converting between formats, and other miscellany regarding peculiarities of the calendar system. It also includes functions for accessing the Lisp Machine's microsecond timer.

Times are represented in two different formats by the functions in the time package. One way is to represent a time by many numbers, indicating a year, a month, a date, an hour, a minute, and a second (plus, sometimes, a day of the week and timezone). If a year less than 100 is specified, a multiple of 100 is added to it to bring it within 50 years of the present. Year numbers returned by the time functions are greater than 1900. The month is 1 for January, 2 for February, etc. The date is 1 for the first day of a month. The hour is a number from 0 to 23. The minute and second are numbers from 0 to 59. Days of the week are fixnums, where 0 means Monday, 1 means Tuesday, and so on. A timezone is specified as the number of hours west of GMT; thus in Massachusetts the timezone is 5. Any adjustment for daylight savings time is separate from this.

This "decoded" format is convenient for printing out times into a readable notation, but it is inconvenient for programs to make sense of these numbers and pass them around as arguments (since there are so many of them). So there is a second representation, called Universal Time, which measures a time as the number of seconds since January 1, 1900, at midnight GMT. This "encoded" format is easy to deal with inside programs, although it doesn't make much sense to look at (it looks like a huge integer). So both formats are provided; there are functions to convert between the two formats; and many functions exist in two versions, one for each format.

The Lisp Machine hardware includes a timer that counts once every microsecond. It is controlled by a crystal and so is fairly accurate. The absolute value of this timer doesn't mean anything useful, since it is initialized randomly; what you do with the timer is to read it at the beginning and end of an interval, and subtract the two values to get the length of the interval in microseconds. These relative times allow you to time intervals of up to an hour (32 bits) with microsecond accuracy.

The Lisp Machine keeps track of the time of day by maintaining a *timebase*, using the microsecond clock to count off the seconds. On the CAIR, when the machine first comes up, the timebase is initialized by querying hosts on the Chaosnet to find out the current time. The Lambda has a calendar clock which never stops, so it normally does not need to do this. You can also set the time base using `time:set-local-time`, described below.

There is a similar timer that counts in 60ths of a second rather than microseconds; it is useful for measuring intervals of a few seconds or minutes with less accuracy. Periodic housekeeping functions of the system are scheduled based on this timer.

34.1 Getting and Setting the Time

get-decoded-time

time:get-time

Gets the current time, in decoded form. Return seconds, minutes, hours, date, month, year, day-of-the-week, and daylight-savings-time-p, with the same meanings as decode-universal-time (see page 782). If the current time is not known, nil is returned.

The name `time:get-time` is obsolete.

get-universal-time

Returns the current time in Universal Time form.

time:set-local-time &optional *new-time*

Sets the local time to *new-time*. If *new-time* is supplied, it must be either a universal time or a suitable argument to `time:parse-universal-time` (see page 781). If it is not supplied, or if there is an error parsing the argument, you are prompted for the new time. Note that you will not normally need to call this function; it is useful mainly when the timebase gets screwed up for one reason or another.

34.1.1 Elapsed Time in 60ths of a Second

The following functions deal with a different kind of time. These are not calendrical date/times, but simply elapsed time in 60ths of a second. These times are used for many internal purposes where the idea is to measure a small interval accurately, not to depend on the time of day or day of month.

time

Returns a number that increases by 1 every 60th of a second. The value wraps around roughly once a day. Use the `time-lessp` and `time-difference` functions to avoid getting in trouble due to the wrap-around. `time` is completely incompatible with the `MacLisp` function of the same name.

Note that `time` with an argument measures the length of time required to evaluate a form. See page 794.

get-internal-run-time

get-internal-real-time

Returns the total time in 60ths of a second since the last boot. This value does not wrap around. Eventually it becomes a bignum. The Lisp Machine does not distinguish between run time and real time.

internal-time-units-per-second

Constant

According to Common Lisp, this is the ratio between a second and the time unit used by values of `get-internal-real-time`. On the Lisp Machine, the value is 60. The value may be different in other Common Lisp implementations.

time-lessp *time1 time2*

t if *time1* is earlier than *time2*, compensating for wrap-around, otherwise nil.

time-difference *time1 time2*

Assuming *time1* is later than *time2*, returns the number of 60ths of a second difference between them, compensating for wrap-around.

time-increment *time interval*

Increments *time* by *interval*, wrapping around if appropriate.

34.1.2 Elapsed Time in Microseconds

time:microsecond-time

Returns the value of the microsecond timer, as a bignum. The values returned by this function wrap around back to zero about once per hour.

time:fixnum-microsecond-time

Returns as a fixnum the value of the low 23 bits of the microsecond timer. This is like `time:microsecond-time`, with the advantage that it returns a value in the same format as the `time` function, except in microseconds rather than 60ths of a second. This means that you can compare `fixnum-microsecond-times` with `time-lessp` and `time-difference`. `time:fixnum-microsecond-time` is also a bit faster, but has the disadvantage that since you only see the low bits of the clock, the value can wrap around more quickly (about every eight seconds). Note that the Lisp Machine garbage collector is so designed that the bignums produced by `time:microsecond-time` are garbage-collected quickly and efficiently, so the overhead for creating the bignums is really not high.

34.2 Printing Dates and Times

The functions in this section create printed representations of times and dates in various formats and send the characters to a stream. To any of these functions, you may pass nil as the *stream* parameter and the function will return a string containing the printed representation of the time, instead of printing the characters to any stream.

The three functions `time:print-time`, `time:print-universal-time`, `time:print-brief-universal-time` and `time:print-current-time` accept an argument called *date-print-mode*, whose purpose is to control how the date is printed. It always defaults to the value of `time:*default-date-print-mode*`. Possible values include:

<code>:dd//mm//yy</code>	Print the date as in '3/16/53'.
<code>:mm//dd//yy</code>	Print as in '16/3/53'.
<code>:dd-mm-yy</code>	Print as in '16-3-53'.
<code>:dd-mmm-yy</code>	Print as in '16-Mar-53'.
<code>: dd mmm yy </code>	Print as in '16 Mar 53'.
<code>:ddmmmyy</code>	Print as in '16Mar53'.

:yymmdd Print as in '530316'.

:yymmmdd Print as in '53Mar16'.

time:print-current-time &optional (*stream* *standard-output*)

Prints the current time, formatted as in 11/25/80 14:50:02, to the specified stream. The date portion may be printed differently according to the argument *date-print-mode*.

time:print-time *seconds minutes hours date month year* &optional
(*stream* *standard-output*) *date-print-mode*

Prints the specified time, formatted as in 11/25/80 14:50:02, to the specified stream. The date portion may be printed differently according to the argument *date-print-mode*.

time:print-universal-time *universal-time* &optional (*stream* *standard-output*)
(*timezone* time:*timezone*) *date-print-mode*

Prints the specified time, formatted as in 11/25/80 14:50:02, to the specified stream. The date portion may be printed differently according to the argument *date-print-mode*.

time:print-brief-universal-time *universal-time* &optional (*stream* *standard-output*)
reference-time *date-print-mode*

This is like *time:print-universal-time* except that it omits seconds and only prints those parts of *universal-time* that differ from *reference-time*, a universal time that defaults to the current time. Thus the output is in one of the following three forms:

02:59	; the same day
3/4 14:01	; a different day in the same year
8/17/74 15:30	; a different year

The date portion may be printed differently according to the argument *date-print-mode*.

time:*default-date-print-mode* *Variable*

Holds the default for the *date-print-mode* argument to each of the functions above. Initially the value here is :mm//dd/yy.

time:print-current-date &optional (*stream* *standard-output*)

Prints the current time, formatted as in Tuesday the twenty-fifth of November, 1980; 3:50:41 pm, to the specified stream.

time:print-date *seconds minutes hours date month year day-of-the-week* &optional
(*stream* *standard-output*)

Prints the specified time, formatted as in Tuesday the twenty-fifth of November, 1980; 3:50:41 pm, to the specified stream.

time:print-universal-date *universal-time* &optional (*stream* *standard-output*)
(*timezone* time:*timezone*)

Prints the specified time, formatted as in Tuesday the twenty-fifth of November, 1980; 3:50:41 pm, to the specified stream.

34.3 Reading Dates and Times

These functions accept most reasonable printed representations of date and time and convert them to the standard internal forms. The following are representative formats that are accepted by the parser. Note that slashes are escaped with additional slashes, as is necessary if these strings are input in traditional syntax.

"March 15, 1960"	"3//15//60"	"3//15//1960"
"15 March 1960"	"15//3//60"	"15//3//1960"
"March-15-60"	"3-15-60"	"3-15-1960"
"15-March-60"	"15-3-60"	"15-3-1960"
"15-Mar-60"	"3-15"	"15 March 60"
"Fifteen March 60"	"The Fifteenth of March, 1960;"	
"Friday, March 15, 1980"		

"1130."	"11:30"	"11:30:17"	"11:30 pm"
"11:30 AM"	"1130"	"113000"	
"11.30"	"11.30.00"	"11.3"	"11 pm"

"12 noon" "midnight" "m" "6:00 gmt" "3:00 pdt"

any date format may be used with any time format

"One minute after March 3, 1960"

meaning one minute after midnight

"Two days after March 3, 1960"

"Three minutes after 23:59:59 Dec 31, 1959"

"Now" "Today" "Yesterday" "five days ago"

"two days after tomorrow" "the day after tomorrow"

"one day before yesterday" "BOB@OZ's birthday"

time:parse *string* &optional (*start* 0) (*end* nil) (*futurep*) *base-time* *must-have-time*
date-must-have-year *time-must-have-second* (*day-must-be-valid*)

Interpret *string* as a date and/or time, and return seconds, minutes, hours, date, month, year, day-of-the-week, daylight-savings-time-p, and relative-p. *start* and *end* delimit a substring of the string; if *end* is nil, the end of the string is used. *must-have-time* means that *string* must not be empty. *date-must-have-year* means that a year must be explicitly specified. *time-must-have-second* means that the second must be specified. *day-must-be-valid* means that if a day of the week is given, then it must actually be the day that corresponds to the date. *base-time* provides the defaults for unspecified components; if it is nil, the current time is used. *futurep* means that the time should be interpreted as being in the future; for example, if the base time is 5:00 and the string refers to the time 3:00, that means the next day if *futurep* is non-nil, but it means two hours ago if *futurep* is nil. The *relative-p* returned value is t if the string included a relative part, such as 'one minute after' or 'two days before' or 'tomorrow' or 'now'; otherwise, it is nil.

If the input is not valid, the error condition `sys:parse-error` is signaled (see page 505).

time:parse-universal-time *string* &optional (*start* 0) (*end* nil) (*future-p* t) *base-time*
must-have-time date-must-have-year time-must-have-second (day-must-be-valid t)

This is the same as `time:parse` except that it returns two values: an integer, representing the time in Universal Time, and the *relative-p* value.

34.4 Reading and Printing Time Intervals

In addition to the functions for reading and printing instants of time, there are other functions specifically for printing time intervals. A time interval is either a number (measured in seconds) or nil, meaning 'never'. The printed representations used look like '3 minutes 23 seconds' for actual intervals, or 'Never' for nil (some other synonyms and abbreviations for 'never' are accepted as input).

time:print-interval-or-never *interval* &optional (*stream* *standard-output*)
interval should be a non-negative fixnum or nil. Its printed representation as a time interval is written onto *stream*.

time:parse-interval-or-never *string* &optional *start end*
 Converts *string*, a printed representation for a time interval, into a number or nil. *start* and *end* may be used to specify a portion of *string* to be used; the default is to use all of *string*. It is an error if the contents of *string* do not look like a reasonable time interval. Here are some examples of acceptable strings:

"4 seconds"	"4 secs"	"4 s"
"5 mins 23 secs"	"5 m 23 s"	"23 SECONDS 5 M"
	"3 yrs 1 week 1 hr 2 mins 1 sec"	
"never"	"not ever"	"no" ""

Note that several abbreviations are understood, the components may be in any order, and case (upper versus lower) is ignored. Also, "months" are not recognized, since various months have different lengths and there is no way to know which month is being spoken of. This function always accepts anything that was produced by `time:print-interval-or-never`; furthermore, it returns exactly the same fixnum (or nil) that was printed.

time:read-interval-or-never &optional (*stream* *standard-input*)
 Reads a line of input from *stream* (using `readline`) and then calls `time:parse-interval-or-never` on the resulting string.

34.5 Time Conversions

decode-universal-time *universal-time* &optional (*timezone* *time:*timezone**)

Converts *universal-time* into its decoded representation. The following values are returned: seconds, minutes, hours, date, month, year, day-of-the-week, daylight-savings-time-p, and the timezone used. *daylight-savings-time-p* tells you whether or not daylight savings time is in effect; if so, the value of *hour* has been adjusted accordingly. You can specify *timezone* explicitly if you want to know the equivalent representation for this time in other parts of the world.

encode-universal-time *seconds minutes hours date month year* &optional *timezone*

Converts the decoded time into Universal Time format, and return the Universal Time as an integer. If you don't specify *timezone*, it defaults to the current timezone adjusted for daylight savings time; if you provide it explicitly, it is not adjusted for daylight savings time. If *year* is less than 100, it is shifted by centuries until it is within 50 years of the present.

time:*timezone*

Variable

The value of *time:*timezone** is the time zone in which this Lisp Machine resides, expressed in terms of the number of hours west of GMT this time zone is. This value does not change to reflect daylight savings time; it tells you about standard time in your part of the world.

34.6 Internal Functions

These functions provide support for those listed above. Some user programs may need to call them directly, so they are documented here.

time:initialize-timebase

Initializes the timebase by querying Chaosnet hosts to find out the current time. This is called automatically during system initialization. You may want to call it yourself to correct the time if it appears to be inaccurate or downright wrong. See also *time:set-local-time*, page 777.

time:daylight-savings-time-p *hours date month year*

Returns *t* if daylight savings time is in effect for the specified hour; otherwise, return *nil*. If *year* is less than 100, it is shifted by centuries until it is within 50 years of the present.

time:daylight-savings-p

Returns *t* if daylight savings time is currently in effect; otherwise, returns *nil*.

time:month-length *month year*

Returns the number of days in the specified *month*; you must supply a *year* in case the month is February (which has a different length during leap years). If *year* is less than 100, it is shifted by centuries until it is within 50 years of the present.

time:leap-year-p *year*

Returns *t* if *year* is a leap year; otherwise return *nil*. If *year* is less than 100, it is shifted by centuries until it is within 50 years of the present.

time:verify-date *date month year day-of-the-week*

If the day of the week of the date specified by *date*, *month*, and *year* is the same as *day-of-the-week*, returns *nil*; otherwise, returns a string that contains a suitable error message. If *year* is less than 100, it is shifted by centuries until it is within 50 years of the present.

time:day-of-the-week-string *day-of-the-week* &optional (*mode*:long)

Returns a string representing the day of the week. As usual, 0 means Monday, 1 means Tuesday, and so on. Possible values of *mode* are:

- :long Returns the full English name, such as "Monday", "Tuesday", etc. This is the default.
- :short Returns a three-letter abbreviation, such as "Mon", "Tue", etc.
- :medium Returns a longer abbreviation, such as "Tues" and "Thurs".
- :french Returns the French name, such as "Lundi", "Mardi", etc.
- :german Returns the German name, such as "Montag", "Dienstag", etc.
- :italian Returns the Italian name, such as "Lunedì", "Martedì", etc.

time:month-string *month* &optional (*mode*:long)

Returns a string representing the month of the year. As usual, 1 means January, 2 means February, etc. Possible values of *mode* are:

- :long Returns the full English name, such as "January", "February", etc. This is the default.
- :short Returns a three-letter abbreviation, such as "Jan", "Feb", etc.
- :medium Returns a longer abbreviation, such as "Sept", "Novem", and "Decem".
- :roman Returns the Roman numeral for *month* (this convention is used in Europe).
- :french Returns the French name, such as "Janvier", "Fevrier", etc.
- :german Returns the German name, such as "Januar", "Februar", etc.
- :italian Returns the Italian name, such as "Gennaio", "Febbraio", etc.

time:timezone-string &optional (*timezone* time:*timezone*)

(*daylight-savings-p* (time:daylight-savings-p))

Return the three-letter abbreviation for this time zone. For example, if *timezone* is 5, then either "EST" (Eastern Standard Time) or "CDT" (Central Daylight Time) is used, depending on *daylight-savings-p*.

ORACLE7™ SERVER SQL LANGUAGE REFERENCE MANUAL

Part Number 776-70-1192
December 1992

ORACLE

Cooperative Server Technology for Transparent Data Sharing

ORACLE7 Server SQL Language Reference Manual

Part No. 778-70-1291

December 1992

Contributing Author: Brian Linden

Contributors: Lori Aster, Steven Bobrowski, Bill Bridge, Stephen Ruff, Gary Hallmark, Michael Harstein, Terry Hayes, Ruth Hillner, Ken Jacobs, Jonathan Klein, Robert Kool, Walter Lindsay, Andrew Mendelsohn, Mark Moore, Ed Peeler, Maria Pratt, and Kevin Wharton

Copyright © 1992 Oracle Corporation. All rights reserved.

This software contains proprietary information of Oracle Corporation. It is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

If this software/documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure of the Program by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

If this software/documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with "Restricted Rights," as defined in FAR 52.227-14, Rights in Data - General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this documentation is error-free.

ORACLE, CASE Dictionary, Pro*Ada, Pro*COBOL, Pro*FORTRAN, Pro*Pascal, Pro*PL/I, SQL*Connect, SQL*DBA, SQL*Forms, SQL*Loader, SQL*Net, and SQL*Plus are registered trademarks of Oracle Corporation.

CASE Designer, CASE Method, ORACLE7, ORACLE Parallel Server, PL/SQL, Pro*C, SQL*Module, and Trusted ORACLE7 are trademarks of Oracle Corporation.

DEC, VAX, and VMS are registered trademarks of Digital Equipment Corporation. DECnet is a trademark of Digital Equipment Corporation.

HP is a registered trademark of Hewlett-Packard.

IBM is a registered trademark of International Business Machines.

DB2 and SQL/DS are trademarks of International Business Machines.

Date Format Elements and National Language Support

The functionality of some date format elements depends on the country and language in which you are using ORACLE. For example, these date format elements return spelled values:

- MONTH
- MON
- DAY
- DY
- BC or AD or B.C. or A.D.
- AM or PM or A.M. or P.M.

The language in which these values are returned is specified either explicitly with the initialization parameter `NLS_DATE_LANGUAGE` or implicitly with the initialization parameter `NLS_LANGUAGE`. The values returned by the YEAR and SYEAR date format elements are always in English.

The date format element D returns the number of the day of the week (1-7). The day of the week that is numbered 1 is specified implicitly by the initialization parameter `NLS_TERRITORY`.

For information on these initialization parameters, see Appendix C "National Language Support" of the *ORACLE7 Server Administrator's Guide*.



Admin

ISO Standard Date Format Elements



Concepts

ORACLE calculates the values returned by the date format elements IYYY, IYY, IY, I, and IW according to the ISO standard. For information on the differences between these values and those returned by the date format elements YYYY, YYY, YY, Y, and WW, see Appendix A "National Language Support" of the *ORACLE7 Server Concepts Manual*.

The RR Date Format Element

The RR date format element is similar to the YY date format element, but it provides additional flexibility for storing date values in other centuries. The RR date format element allows you to store twenty-first century dates in the twentieth century by specifying only the last two digits of the year. It will also allow you to store twentieth century dates in the twenty-first century in the same way if necessary.

If you use the `TO_DATE` function with the YY date format element, the date value returned is always in the current century. If you use the RR date format element instead, the century of the return value varies according to the specified two-digit year and the last two digits of the current year. Table 3-16 summarizes the behavior of the RR date format element.

TABLE 3-14
The RR Date Format Element

		If the specified two-digit year is	
		00-49	50-99
If the last two digits of the current year are:	00-49	The return date is in the current century.	The return date is in the century before the current one.
	50-99	The return date is in the century after the current one.	The return date is in the current century.

The following example demonstrates the behavior of the RR date format element.

Example IV Assume these queries are issued prior to the year 2000:

```
SELECT TO_CHAR(TO_DATE('27-OCT-95', 'DD-MON-RR'), 'YYYY') "4-digit year"
FROM DUAL;

4-digit year
-----
1995
```

```
SELECT TO_CHAR(TO_DATE('27-OCT-17', 'DD-MON-RR'), 'YYYY') "4-digit year"
FROM DUAL;

4-digit year
-----
2017
```

Assume these queries are issued in the year 2000 or after:

```
SELECT TO_CHAR(TO_DATE('27-OCT-95', 'DD-MON-RR'), 'YYYY') "4-digit year"
FROM DUAL;

4-digit year
-----
1995
```

```
SELECT TO_CHAR(TO_DATE('27-OCT-17', 'DD-MON-RR'), 'YYYY') "4-digit year"
FROM DUAL;

4-digit year
-----
2017
```

Note that the queries return the same values regardless of whether they are issued before or after the year 2000. The RR date format element allows you to write SQL statements that will return the same values after the turn of the century.

Server Products

ORACLE[®] Server

SOL Language Reference Manual

Oracle

#13

1
⑥

INTERNATIONAL STANDARD

ISO
8601

First edition
1988-06-15



INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ

**Data elements and interchange formats —
Information interchange — Representation of dates
and times**

*Éléments de données et formats d'échange — Échange d'information — Représentation de
la date et de l'heure*

Reference number
ISO 8601:1988 (E)

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council. They are approved in accordance with ISO procedures requiring at least 75 % approval by the member bodies voting.

International Standard ISO 8601 was prepared by Technical Committee ISO/TC 154, *Documents and data elements in administration, commerce and industry*.

It cancels and replaces International Standards ISO 2014 : 1976, ISO 2015 : 1976, ISO 2711 : 1973, ISO 3307 : 1975 and ISO 4031 : 1978, of which it constitutes a technical revision.

Users should note that all International Standards undergo revision from time to time and that any reference made herein to any other International Standard implies its latest edition, unless otherwise stated.

Contents

	Page
0 Introduction	1
1 Scope and field of application	1
2 References	2
3 Terms and definitions	2
4 Fundamental principles	3
4.1 Concept	3
4.2 Common features, uniqueness and combinations	3
4.3 Characters used in the representations	3
4.4 Use of separators	3
4.5 Truncation	3
4.6 Leading zero(s)	3
5 Representations	3
5.1 Explanations	3
5.2 Dates	4
5.3 Time of day	6
5.4 Combinations of date and time of the day representations	7
5.5 Periods of time	8
Annexes	
A Relationship to ISO 2014, 2015, 2711, 3307 and 4031	10
B Examples of representation of dates, time of the day, combinations of date and time, and periods of time	11

Data elements and interchange formats — Information interchange — Representation of dates and times

0 Introduction

0.1 Although ISO Recommendations and Standards in this field have been available since 1971, different forms of numeric representation of dates and times have been in common use in different countries. Where such representations are interchanged across national boundaries misinterpretation of the significance of the numerals can occur, resulting in confusion and other consequential errors or losses. The purpose of this International Standard is to eliminate the risk of misinterpretation and to avoid the confusion and its consequences.

0.2 This International Standard includes specifications for the numeric representation of information regarding date and time of the day.

0.3 In order to achieve similar formats for the representations of calendar dates, ordinal dates, dates identified by week number, periods of time, combined date and time of the day, and differences between local time and Coordinated Universal Time, and to avoid ambiguities between these representations, it has been necessary to use, apart from numeric characters, either single alphabetic characters or one or more other graphic characters or a combination of alphabetic and other characters in some of the representations.

0.4 The above action has had the benefit of enhancing the versatility and general applicability of previous International Standards in this field, and provides for the unique representation of any date or time expression or combination of these. Each representation can be easily recognized, which is beneficial when human interpretation is required.

0.5 This International Standard retains the most commonly used expressions for date and time of the day and their representations from the earlier International Standards and provides unique representations for some new expressions used in practice. Its application in information interchange, especially between data processing systems and associated equipment will eliminate errors arising from misinterpretation and the costs these generate. The promotion of this Inter-

national Standard will not only facilitate interchange across international boundaries, but will also improve the portability of software, and will ease problems of communication within an organization, as well as between organizations.

0.6 Several of the alphabetic and graphic characters used in the text of this International Standard are common both to the representations specified and to normal typographical presentation.

0.7 To avoid confusion between the representations and the actual text, its punctuation marks and associated graphic characters, all the representations are contained in brackets []. The brackets are not part of the representation, and should be omitted when implementing the representations. All matter outside the brackets is normal text, and not part of the representation. In the associated examples, the brackets and typographical markings are omitted.

1 Scope and field of application

This International Standard specifies the representation of dates in the Gregorian calendar and times and representations of periods of time. It includes

- a) calendar dates expressed in terms of year, month and day of month;
- b) ordinal dates expressed in terms of year and day of year;
- c) dates identified by means of year, week numbers and day numbers;
- d) time of the day based upon the 24-hour timekeeping system;
- e) differences between local time and Coordinated Universal Time (UTC);
- f) combination of date and time;
- g) periods of time, with or without either a start or end date or both.

ISO 8601 : 1988 (E)

This International Standard is applicable whenever dates and times are included in information interchange.

This International Standard does not cover dates and times where words are used in the representation.

This International Standard does not assign any particular meaning or interpretation to any data element that uses representations in accordance with this International Standard. Such meaning will be determined by the context of the application.

2 References

ISO 31-0 : 1981, *General principles concerning quantities, units and symbols*.

ISO 31-1 : 1978, *Quantities and units of space and time*.

ISO 646 : 1983, *Information processing — ISO 7-bit coded character set for information interchange*.

3 Terms and definitions

For the purposes of this International Standard, the following terms and definitions apply.

3.1 complete representation: The representation that includes all the date and time elements associated with the expression.

3.2 Coordinated Universal Time (UTC): The time scale maintained by the Bureau International de l'Heure (International Time Bureau) that forms the basis of a coordinated dissemination of standard frequencies and time signals.

NOTES

1 The source of this definition is Recommendation 460-2 of the Consultative Committee on International Radio (CCIR). CCIR has also defined the acronym for Coordinated Universal Time as UTC (see also 5.3.3).

2 UTC is often (incorrectly) referred to as Greenwich Mean Time and appropriate time signals are regularly broadcast.

3.3 date, calendar: A particular day of a calendar year, identified by its ordinal number within a calendar month within that year.

3.4 date, ordinal: A particular day of a calendar year identified by its ordinal number within the year.

3.5 day: A period of time of 24 hours starting at 0000 and ending at 2400 (which is equal to the beginning of 0000 the next day).

3.6 format, basic: The format of a representation comprising the minimum number of components necessary for the precision required.

3.7 format, extended: An extension of the basic format that includes additional separators.

3.8 Gregorian calendar: A calendar in general use introduced in 1582 to correct an error in the Julian calendar. In the Gregorian calendar common years have 365 days and leap years 366 days divided into 12 sequential months.

3.9 hour: A period of time of 60 minutes.

3.10 local time: The clock time in public use locally.

3.11 minute: A period of time of 60 seconds.

3.12 month, calendar: A period of time resulting from the division of a calendar year in twelve sequential periods of time, each with a specific name and containing a specified number of days. In the Gregorian calendar, the months of the calendar year, listed in their order of occurrence, are named and contain the number of days as follows: January (31), February (28 in common years; 29 in leap years), March (31), April (30), May (31), June (30), July (31), August (31), September (30), October (31), November (30), December (31).

NOTE — In certain applications a month is regarded as a period of 30 days.

3.13 period: A duration of time, specified

- a) as a defined length of time (e.g. hours, days, months, years);
- b) by its beginning and end points.

3.14 second: A basic unit of measurement of time in the International System of Units (SI) as defined in ISO 31-1.

3.15 truncated representation: The abbreviation of a complete representation by omission of higher order components starting from the extreme left-hand side of the expression.

3.16 week: A period of time of seven days.

3.17 week, calendar: A seven day period within a calendar year, starting on a Monday and identified by its ordinal number within the year; the first calendar week of the year is the one that includes the first Thursday of that year. In the Gregorian calendar, this is equivalent to the week which includes 4 January.

3.18 year: A period of time of twelve consecutive months, considered to equal a calendar year.

3.19 year, calendar: A cyclic period of time in a calendar which is required for one revolution of the earth around the sun. In the Gregorian calendar, a calendar year is either a common year or a leap year.

3.20 year, common: In the Gregorian calendar, a year which has 365 days.

3.21 year, leap: In the Gregorian calendar, a year which has 366 days. A leap year is a year whose number is divisible by four an integral number of times, except that if it is a centennial year it shall be divisible by four hundred an integral number of times.

4 Fundamental principles

4.1 Concept

A precise point in calendar time can be identified by means of a unique expression giving a specific day and a specific time within that day. The degree of precision required for the application can be obtained by including the appropriate components.

4.2 Common features, uniqueness and combinations

The decreasing order of components, left-to-right, is common to the expressions for

- precise points in time;
- dates only;
- times only;
- periods of time;
- any abbreviations of the above.

4.3 Characters used in the representations

The representations specified in this International Standard use digits, alphabetic characters and special characters specified in ISO 646. The particular use of these characters is explained in 4.4 and clause 5.

NOTE — Where the upper case characters are not available lower case characters may be used.

The space character shall not be used in the representations.

4.4 Use of separators

When required, the following characters shall be used as separators:

[-] (hyphen) — to separate the time elements "year" and "month", "year" and "week", "year" and "day", "month" and "day", and "week" and "day";

NOTE — The hyphen is also used to indicate omitted components.

[:] (colon) — to separate the time elements "hour" and "minute", and "minute" and "second".

[/] (solidus) — to separate the two components in the representation of periods of time.

4.5 Truncation

It is permitted to omit higher order components (truncation) in applications where their presence is implied. To assure uniqueness of each representation provided for in this International Standard, truncation of a particular representation should be done in accordance with the rules given in the appropriate subclause of clause 5 referring to the representation

in question. The addition of a single hyphen in place of each omitted component will usually be necessary, to avoid risk of misinterpretation.

NOTE — By mutual agreement of the partners in information interchange, leading hyphens may be omitted in the applications where there is no risk of confusing these representations with others defined in this International Standard.

4.6 Leading zero(s)

Each date and time component in a defined representation has a defined length, and (a) leading zero(s) shall be used as required.

5 Representations

5.1 Explanations

5.1.1 Characters used in place of digits

[C] represents a digit used in the thousands and hundreds components (the "century" component) of the time element "year";

[Y] represents a digit used in the tens and units components of the time element "year";

[M] represents a digit used in the time element "month";

[D] represents a digit used in the time element "day";

[w] represents a digit used in the time element "week";

[h] represents a digit used in the time element "hour";

[m] represents a digit used in the time element "minute";

[s] represents a digit used in the time element "second";

[n] represents digit(s), constituting a positive integer.

5.1.2 Characters used as designators

[P] is used as period designator, preceding a data element which represents a given duration of a period of time;

[T] is used as time designator to indicate the start of the representation of the time of the day in combined date and time of day expressions;

[W] is used as week designator, preceding a data element which represents the ordinal number of a calendar week within the year;

[Z] is used as time-zone designator, immediately (without space) following a data element expressing the time of the day in Coordinated Universal Time (UTC).

In representations of duration of time (5.5.3.2), the following characters are also used as parts of the representation when required:

[Y] [M] [W] [D] [H] [M] [S]

NOTE — In these representations, [M] may be used to indicate "month" or "minute", or both.

5.2 Dates

For ease of comparison, in all the following examples of representations of dates, the date of 12 April 1985 is used as an illustration, as applicable.

5.2.1 Calendar date

In expressions of calendar dates

- day of the month (calendar day) is represented by two digits. The first day of any month is represented by [01] and subsequent days of the same month are numbered in ascending sequence;
- month is represented by two digits. January is represented by [01], and subsequent months are numbered in ascending sequence;
- year is generally represented by four digits; years are numbered in ascending order according to the Gregorian Calendar.

5.2.1.1 Complete representation

When the application clearly identifies the need for an expression only of a calendar date, then the complete representation shall be a single numeric data element comprising eight digits, where [CCYY] represents a calendar year, [MM] the ordinal number of a calendar month within the calendar year, and [DD] the ordinal number of a day within the calendar month.

Basic format: CCYYMMDD

Example: 19850412

Extended format: CCYY-MM-DD

Example: 1985-04-12

5.2.1.2 Representations with reduced precision

If in a given application it is sufficient to express a calendar date with less precision than a complete representation as specified in 5.2.1.1, either two, four or six digits may be omitted, the omission starting from the extreme right-hand side. The resulting representation will then indicate a month, a year or a century, as set out below. When only [DD] are omitted, a separator shall be inserted between [CCYY] and [MM], but separators shall not be used in the other representations with reduced precision.

a) A specific month

Basic format: CCYY-MM

Example: 1985-04

Extended format: not applicable

b) A specific year

Basic format: CCYY

Example: 1985

Extended format: not applicable

c) A specific century

Basic format: CC

Example: 19

Extended format: not applicable

5.2.1.3 Truncated representations

If truncated representations are required the basic formats shall be as specified below. In each case hyphens (to indicate omitted components) shall be used only as indicated.

a) A specific date in the current century

Basic format: YYMMDD

Example: 850412

Extended format: YY-MM-DD

Example: 85-04-12

b) A specific year and month in the current century

Basic format: -YYMM

Example: -8504

Extended format: -YY-MM

Example: -85-04

c) A specific year in the current century

Basic format: -YY

Example: -85

Extended format: not applicable

d) A specific day of a month

Basic format: --MMDD

Example: --0412

Extended format: --MM-DD

Example: --04-12

e) A specific month

Basic format: --MM

Example: --04

Extended format: not applicable

f) A specific day

Basic format: ---DD

Example: ---12

Extended format: not applicable

5.2.2 Ordinal date

The ordinal day of the year is represented by three decimal digits. The first day of any year is represented by [001] and subsequent days are numbered in ascending sequence.

5.2.2.1 Complete representation

When the application clearly identifies the need for a complete representation of an ordinal date, it shall be one of the numeric

expressions as follows, where [CCYY] represents a calendar year and [DDD] the ordinal number of a day within the year.

Basic format: CCYYDDD

Example: 1985102

Extended format: CCYY-DDD

Example: 1985-102

5.2.2.2 Truncated representations

If truncated representations are required, the basic formats shall be as specified below. In each case hyphens (to indicate omitted components) shall be used only as indicated.

- a) A specific year and day in the current century

Basic format: YYDDD

Example: 85102

Extended format: YY-DDD

Example: 85-102

- b) Day only

Basic format: -DDD

Example: -102

Extended format: not applicable

NOTE — Logically, the representation should be [-DDD], but the first hyphen is superfluous and, therefore, it has been omitted.

5.2.3 Date identified by calendar week and day numbers

Calendar week is represented by two numeric digits. The first calendar week of a year shall be identified as [01] and subsequent weeks shall be numbered in ascending sequence.

Day of the week is represented by one decimal digit. Monday shall be identified as day [1] of any calendar week, and subsequent days of the same week shall be numbered in ascending sequence to Sunday (day [7]).

5.2.3.1 Complete representation

When the application clearly identifies the need for a complete representation of a date identified by calendar week and day numbers, it shall be one of the alphanumeric expressions as follows, where [CCYY] represents a calendar year, [W] is the week designator, [ww] represents the ordinal number of a calendar week within the year, and [D] represents the ordinal number of a day within the calendar week.

Basic format: CCYYWwwwD

Example: 1985W155

Extended format: CCYY-Wwww-D

Example: 1985-W15-5

5.2.3.2 Representation with reduced precision

If the degree of precision required permits, one digit may be omitted from the representation in 5.2.3.1.

Basic format: CCYYWwww

Example: 1985W15

Extended format: CCYY-Wwww

Example: 1985-W15

5.2.3.3 Truncated representations

If truncated representations are required the basic formats shall be as specified below. In each case hyphens (to indicate omitted components) shall be used only as indicated.

- a) Year, week and day in the current century

Basic format: YYWwwwD

Example: 85W155

Extended format: YY-Wwww-D

Example: 85-W15-5

- b) Year and week only in the current century

Basic format: YYWwww

Example: 85W15

Extended format: YY-Wwww

Example: 85-W15

- c) Year of the current decade, week and day only

Basic format: -YWwwwD

Example: -5W155

Extended format: -Y-Wwww-D

Example: -5-W15-5

- d) Week and day only of the current year

Basic format: -WwwwD

Example: -W155

Extended format: -Wwww-D

Example: -W15-5

- e) Week only of the current year

Basic format: -Wwww

Example: -W15

Extended format: not applicable

- f) Day only of the current week

Basic format: -W-D

Example: -W-5

Extended format: not applicable

NOTE — Although the representation [-W-D] could be abbreviated to [-D] without risk of misinterpretation, the full, logical, derivation

has been retained because the [W] serves to identify the representation as a date based on week and day numbers. Its frequency of use is expected to be low and, therefore, the two potentially superfluous characters are not likely to create transmission problems.

g) Day only of any week

Basic format: --D

Example: --5

Extended format: not applicable

5.3 Time of the day

As this International Standard is based on the 24-hour timekeeping system which is now in common use, hours are represented by two digits from [01] to [24], whereas minutes and seconds are represented by two digits from [01] to [60]. For most purposes times will be represented by four digits [hhmm].

5.3.1 Local time of the day

5.3.1.1 Complete representation

When the application clearly identifies the need for an expression only of a time of the day then the complete representation shall be a single numeric data element comprising six digits in the basic format, where [hh] represents hours, [mm] minutes and [ss] seconds.

Basic format: hhmmss

Example: 232050

Extended format: hh:mm:ss

Example: 23:20:50

5.3.1.2 Representations with reduced precision

If the degree of precision required permits, either two or four digits may be omitted from the representation in 5.3.1.1.

Basic format: hhmm
hh

Example: 2320
23

Extended format: hh:mm
not applicable

Example: 23:20

5.3.1.3 Representation of decimal fractions

If necessary for a particular application a decimal fraction of hour, minute or second may be included. If a decimal fraction is included, lower order components (if any) shall be omitted, and the decimal fraction shall be divided from the integer part by the decimal sign specified in ISO 31-0: i.e. the comma [,] or full stop [.]. Of these, the comma is the preferred sign. If the magnitude of the number is less than unity, the decimal sign shall be preceded by a zero (see ISO 31-0).

The number of digits in the decimal fraction shall be determined by the interchange parties, dependent upon the application. The format shall be [hhmmss,s], [hhmm,m] or [hh,h] as appropriate (hour minute second, hour minute and hour, respectively), with as many digits as necessary following the decimal sign. If the extended format is required, separators may be included in the decimal representation when the complete representation is used, or when it is reduced by omission of [ss,s].

Basic format: hhmmss,s
hhmm,m
hh,h

Example: 232050,5
2320,9
23,3

Extended format: hh:mm:ss,s
hh:mm,m
not applicable

Example: 23:20:50,5
23:20,9

5.3.1.4 Truncated representations

If truncated representations are required the basic formats shall be as specified below. In each case hyphens (to indicate omitted components) shall be used only as indicated.

a) A specific minute and second of the hour

Basic format: -mmss

Example: -2050

Extended format: -mm:ss

Example: -20:50

b) A specific minute of the hour

Basic format: -mm

Example: -20

Extended format: not applicable

c) A specific second of the minute

Basic format: --ss

Example: --50

Extended format: not applicable

d) A specific hour of the day and decimal fraction of the hour

Basic format: hh,h

Example: 11,3

Extended format: not applicable

e) A specific minute of the hour and a decimal fraction of the minute

Basic format: -mm,m

Example: -20,9

Extended format: not applicable

- f) A specific minute and second of the hour and a decimal fraction of the second

Basic format: -mmss,s

Example: -2050,5

Extended format: -mm:ss,s

Example: -20:50,5

- g) A specific second of the minute and a decimal fraction of the second

Basic format: -ss,s

Example: -50,5

Extended format: not applicable

NOTE — The basic formats above show only one digit following the decimal sign, but as many digits as required may be used.

5.3.2 Midnight

The complete and extended representations for midnight, in accordance with 5.3.1, shall be expressed in either of the two following ways:

<i>Basic format</i>	<i>Extended format</i>
a) 000000	00:00:00 (the beginning of a day);
b) 240000	24:00:00 (the end of a day).

The representations may be reduced in accordance with 5.3.1.4.

NOTES

- 1 Midnight will normally be represented as [0000] or [2400]
- 2 The choice of representation a) or b) will depend upon any association with a date, or a time period.
- 3 The end of one day (2400) coincides with (0000) at the start of the next day, e.g. 2400 on 12 April 1985 is the same as 0000 on 13 April 1985. If there is no association with a date or a time period both a) and b) represent the same clock time in the 24-hour timekeeping system.

5.3.3 Coordinated Universal Time (UTC)

To express the time of the day in Coordinated Universal Time, the representations specified in 5.3.1 shall be used, followed immediately, without spaces, by the time-zone designator [Z]. The examples below are complete and reduced precision representations of the UTC time 20 minutes and 30 seconds past 23 hours:

Basic format: hhmmssZ
hhmmZ
hhZ

Example: 232030Z
2320Z
23Z

Extended format: hh:mm:ssZ
hh:mmZ
not applicable

Example: 23:20:30Z
23:20Z

5.3.3.1 Differences between local time and Coordinated Universal Time

When it is required to indicate the difference between local time and Coordinated Universal Time, its representation shall be appended to the representation of the local time following immediately, without space, the lowest order (extreme right-hand) component of the local time expression, which, in this case, shall always include hours.

The difference between local time and Coordinated Universal Time shall be expressed in hours and minutes, or hours only independently of the precision of the local time expression associated with it. It shall be expressed as positive (i.e. with the leading plus sign [+]) if the local time is ahead of and as negative (i.e. with the leading minus sign [-]) if it is behind Coordinated Universal Time as shown below. The complete representation of the time of 27 minutes 46 seconds past 15 hours locally in Geneva (normally one hour ahead of UTC), and in New York (five hours behind UTC), together with the indication of the difference between the local time and Coordinated Universal Time, are used as examples.

Basic format: +hhmm
+hh
-hhmm
-hh

Example: 152746+0100
152746+01
152746-0500
152746-05

Extended format: +hh:mm
not applicable
-hh:mm
not applicable

Example: 15:27:46+01:00
15:27:46+01
15:27:46-05:00
15:27:46-05

NOTE — The representations of the negative difference between local time and Coordinated Universal Time should not be used alone as they may be confused with the truncated representations of dates provided for in 5.2.1.3, and with truncated representations of time of the day provided for in 5.3.1.4.

5.4 Combinations of date and time of the day representations

When the application does not clearly identify the need for only a date expression (see 5.2) or only a time of the day expression (see 5.3), then a moment of time can be identified through a combination of date and time of the day representations provided for in this International Standard.

5.4.1 Complete representation

The components of an instant of time shall be written in the following sequence:

- a) For calendar dates:
- year - month - day - time designator - hour - minute - second

ISO 8601 : 1988 (E)

b) For ordinal dates:

year - day - time designator - hour - minute - second

c) For dates identified by week and day numbers:

year - week designator - week - day - time designator -
hour - minute - second

The character [T] shall be used as time designator to indicate the start of the representation of date time of day in combined date and time of day expressions. The hyphen [-] and the colon [:] shall be used, in accordance with 4.4, as separators within the date and time of the day expressions respectively, when required. When any of the date or time components are omitted, the time designator shall always precede the remaining time of day components.

NOTE — By mutual agreement of the partners in information interchange, the character [T] may be omitted in applications where there is no risk of confusing a combined date and time of the day representation with others defined in this International Standard.

The following are examples of complete and reduced representation (in basic and extended format) of combinations of date and time of the day representations:

a) Calendar date and local time of the day

Basic format: CCYYMMDDThhmmss
CCYYMMDDThhmm
CCYYMMDDThh

Examples: 19850412T101530
19850412T1015
19850412T10

Extended format: CCYY-MM-DDThh:mm:ss
CCYY-MM-DDThh:mm
CCYY-MM-DDThh

Examples: 1985-04-12T10:30
1985-04-12T10:15
1985-04-12T10

b) Ordinal date and local time of the day

Basic format: CCYYDDDDThhmmss
CCYYDDDDThhmm
CCYYDDDDThh

Examples: 1985102T235030
1985102T2350
1985102T23

Extended format: CCYY-DDDDThh:mm:ss
CCYY-DDDDThh:mm
CCYY-DDDDThh

Examples: 1985-102T23:50:30
1985-102T23:50
1985-102T23

c) Date identified by calendar week and day numbers and local time of the day

Basic format: CCYYWwwwDThhmmss
CCYYWwwwDThhmm
CCYYWwwwDThh

Examples: 1985W155T235030
1985W155T2350
1985W155T23

Extended format: CCYY-Wwww-DThh:mm:ss
CCYY-Wwww-DThh:mm
CCYY-Wwww-DThh

Examples: 1985-W15-5T23:30
1985-W15-5T23:50
1985-W15-5T23

5.4.2 Representations other than complete

For reduced precision or truncated representations of a combined date and time expression any of the representations in 5.2.1 (for calendar dates), 5.2.2 (for ordinal dates) or 5.2.3 (for dates identified by week numbers) may be combined with any of the representations in 5.3 provided that the rules specified in those sections are applied, together with the following:

- a) the date component shall not be represented with reduced precision and the time component shall not be truncated in a combined date and time expression;
- b) when truncation occurs in the date component of a combined date and time expression, it is not necessary to replace the omitted higher order components with the hyphen [-];
- c) when the context does not clearly identify a time only component, and if the extended format including colon [:] separator is not used, then it is necessary to commence the time expression with the designator [T].

5.5 Periods of time

5.5.1 Means of specifying periods

A period of time shall be expressed in one of the following ways:

- a) As a duration of time delimited by a specific start and a specific end;
- b) As a quantity of time expressed in one or more specific components but not associated with any specific start or end;
- c) As a quantity of time associated with a specific start;
- d) As a quantity of time associated with a specific end.

5.5.2 Separators and designators

A solidus [/] shall be used to separate the two components in each of 5.5.1 a), c) and d).

For 5.5.1 b), c) and d) the designator [P] shall precede, without spaces, the representation of the duration.

Other designators (and the hyphen when used to indicate omitted components) shall be used as shown in 5.5.3 below.

NOTE — In certain application areas a double hyphen is used as a separator instead of a solidus.

5.5.3 Complete representations

5.5.3.1 Representation of period identified by its start and end

When the application clearly identifies the need for a complete representation of a period of time, identified by its start and its end, it shall be one of the alphanumeric expressions as set out below. For the specific start or end of a period, [CCYY] represents a calendar year, [MM] the ordinal number of a calendar month within the calendar year, [DD] the ordinal number of a day within the calendar month, [hh] hours, [mm] minutes and [ss] seconds.

Basic format:

CCYYMMDDThhmmss/CCYYMMDDThhmmss

Example: 19850412T232050/19850625T103000

A period beginning at 20 minutes and 50 seconds past 23 hours on 12 April 1985 and ending at 30 minutes past 10 hours on 25 June 1985.

5.5.3.2 Representation of duration of time

A given duration of a period of time, whether or not associated with a start or end, shall be represented by a data element of variable length, preceded by the designator [P]. The number of years shall be followed by the designator [Y], the number of months by [M], the number of weeks by [W], and the number of days by [D]. The part including time components shall be preceded by the designator [T]; the number of hours shall be followed by [H], the number of minutes by [M] and the number of seconds by [S]. In the example set out below, [n] represents one or more digits, constituting a positive integer.

Basic format: PnYnMnDnHnMnS

PnW

Example: P2Y10M15DT10H30M20S

A duration of two years, 10 months, 15 days, 10 hours, 30 minutes and 20 seconds.

P6W

A period of six weeks.

5.5.3.2.1 Alternative format

If required for particular reasons, durations of time may be expressed in conformity with the format used for points-in-time, as specified in clause 5. Accordingly, the values expressed must not exceed the "carry-over points" of 12 months, 30 days, 24 hours, 60 minutes and 60 seconds. Since weeks have no defined carry-over point (52 or 53), weeks should not be used in these applications.

5.5.3.3 Representation of period identified by its start and its duration

Basic format:

CCYYMMDDThhmmss/PnYnMnDnHnMnS

Example: 19850412T232050/P1Y2M15DT12H30M

A period of one year, 2 months, 15 days, 12 and a half hours, beginning on 12 April 1985 at 20 minutes and 50 seconds past 23 hours.

5.5.3.4 Representation of period identified by its duration and its end

Basic format:

PnYnMnDnHnMnS/CCYYMMDDThhmmss

Example: P1Y2M15DT12H30M/19850412T232050

A period of one year, 2 months, 15 days and 12 and a half hours, ending on 12 April 1985 at 20 minutes and 50 seconds past 23 hours.

NOTES

1 Where complete representations using calendar dates have been shown, ordinal dates (5.2.2) or dates identified by week number (5.2.3) may be substituted in similar fashion.

2 In 5.5.3.2, 5.5.3.3 and 5.5.3.4 the components for duration would frequently be in reduced precision form.

If extended formats are required, they shall conform to the requirements of 5.2.1.1, 5.2.2.1, 5.2.3.1 and 5.3.1.1.

5.5.4 Representations other than complete

If reduced precision, or truncated, or decimal representations, or extended formats, are used in place of any components in the complete representations, they shall each be in accordance with the corresponding rules in 5.2 and 5.3.

In representation for the periods in 5.5.1 a),

— if higher order components are omitted from the expression following the solidus (i.e. the representation for "end of period"), it shall be assumed that the corresponding components from the "start of period" expression apply (e.g. if [CCYYMM] are omitted by using a derived representation, the end of the period is in the same year and month as the start of the period);

— representations for time-zones and Coordinated Universal Time included with the component preceding the solidus shall be assumed to apply to the component following the solidus, unless a corresponding alternative is included.

Annex A

Relationship to ISO 2014, 2015, 2711, 3307 and 4031

(This annex does not form part of the standard.)

A.1 In preparing the first edition of ISO 2014 an examination was carried out of the potential uses of all-numeric dates. The advantages of the descending order year-month-day were found to outweigh those for the ascending order day-month-year already established at that time in many parts of the world.

The advantages of the descending order were found to include the following, in particular:

- a) the avoidance of confusion in comparison with existing national conventions using different systems of ascending order;
- b) the ease with which the whole date may be treated as a single numeral for the purposes of filing and classification;
- c) arithmetic calculation, particularly in some computer uses;
- d) the possibility of continuing the order by adding digits for hour-minute-second.

A.2 For times, use of the 24-hour timekeeping system is now so common (particularly in view of the wide availability and use of digital watches) that separators to aid human interpretation are no longer necessary but are included as options.

The natural addition of the lower order time digits to the higher order date digits (see above) established the basic concept used in the preparation of this International Standard: that a point in time could be uniquely represented in all-numeric form by a string of digits commencing with year and ending with hour, minute or second, depending on the precision desired.

From that concept representations of all other date and time values were logically derived and, thus, ISO 2014, ISO 3307 and ISO 4031 have been superseded.

A.3 Numbering of days and weeks in the year based on the Gregorian calendar is important in many commercial applications. Methods of numbering the weeks of the year vary from country to country, and, therefore, for international trade and for industrial planning within international companies it is essential to use uniform numbering of weeks. ISO 2015 and ISO 2711 were prepared to meet these requirements.

The uniform numbering of weeks necessitates a unique designation of the day on which a week begins. For commercial purposes, i.e. accounting, planning and similar purposes for which a week number might be used, Monday has been found the most appropriate as the first day of the week.

Identification of a particular date by means of ordinal dates (ISO 2711) and by means of the week numbering system (ISO 2015) were alternative methods that the basic concept of this International Standard could also encompass and, thus, ISO 2015 and ISO 2711 have now been superseded.

Annex B

Examples of representation of dates, time of the day, combinations of date and time, and periods of time

(This annex does not form part of the standard.)

B.1 Dates

Basic format	Extended format	Explanations
Calendar date — 12 April 1985		
19850412	1985-04-12	Complete
850412	85-04-12	Year of any century, with month and date only
--0412	-04-12	Month and date of any year
---12	not applicable	Day only of any month
Ordinal date — 12 April 1985		
1985102	1985-102	Complete
85102	85-102	Year of any century, with ordinal day
5-102	not applicable	Year of any decennium, with ordinal day
-102	not applicable	Ordinal day of any year
Calendar week and day — Friday 12 April 1985		
1985W155	1985-W15-5	Complete
85W155	85-W15-5	Year of any century, with week and day
-5W155	-5-W15-5	Year of any decennium, with week and day
-W155	-W15-5	Week and day of any year
-W-5	not applicable	Any week and day of that week
Calendar week — 15th week of 1985		
1985W15	1985-W15	Complete
85W15	85-W15	Year of any century and week of that year
-5W15	-5W15	Year of any decennium and week of that year
-W15	not applicable	Specific week of any year
Day of the week — Friday		
--5	not applicable	Any Friday
Calendar month — April 1985		
1985-04	not applicable	Complete
-8504	-85-04	Year of any century and month of that year
-04	not applicable	Specific month of any year
Calendar year — 1985		
1985	not applicable	Complete
-85	not applicable	Specific year of any century

ISO 8601 : 1988 (E)

B.2 Time of the day

Basic format	Extended format	Explanations
Local time of the day		
27 minutes 46 seconds past 15 hours locally		
152746	15:27:46	Complete
-2746	-27:46	Specific minute and second of any hour
--46	not applicable	Specific second of any minute
Reduced to hours and minutes		
1527	15:27	Complete
-27	not applicable	Specific minute of any hour
Reduced to hours		
15	not applicable	Specific hour of any day
Local time with decimal fractions		
27 minutes 35 and a half seconds past 15 hours locally		
152735,5	15:27:35,5	Complete
-2735,5	-27:35,5	Minute of hour, second with decimal fraction
--35,5	not applicable	Second with decimal fraction of the minute
15,46	not applicable	Hour with decimal fraction of that hour
-27,59	not applicable	Minute with decimal fraction of that minute
-,59	not applicable	Decimal fraction of the minute
-,5	not applicable	Decimal fraction of the second
Midnight — The beginning of a day		
000000	00:00:00	Complete
0000	00:00	Hour and minute only
Midnight — The end of the day		
240000	24:00:00	Complete
2400	24:00	Hour and minute only
Coordinated Universal Time (UTC)		
20 minutes and 30 seconds past 23 hours UTC		
232030Z	23:20:30Z	Complete
2320Z	23:20Z	Hour and minute in UTC
23Z	not applicable	Hour in UTC
Differences between local time and Coordinated Universal Time		
The time of 27 minutes 46 seconds past 15 hours locally in Geneva (one hour ahead of UTC)		
152746 + 0100	15:27:46 + 01:00	Complete
152746 + 01	15:27:46 + 01	Time difference expressed in hours only
The same time locally in New York (five hours behind UTC)		
152746-0500	15:27:46-05:00	Complete
152746-05	15:27:46-05	Time difference expressed in hours only

B.3 Combinations of date and time

Basic format	Extended format	Explanations
Combinations of calendar date and local time of the day		
19850412T101530	1985-04-12T10:15:30	Complete
850412T101530	85-04-12T10:15:30	Within specific year of any century
850412T1015	85-04-12T10:15	Ditto, with hour and minute only
0412T1015	04-12T10:15	Within specific month of any year, with hour and minute only
0412T10	04-12T10	Ditto, with hour only
12T10	12T10	Within specific day of any month, with hour only
850412T10	85-04-12T10	Within specific date of any century, with hour only
12T101530	12T10:15:30	Within specific day of any month, year and century
etc.		
Combinations of ordinal date and local time of the day		
1985102T235030	1985-102T23:50:30	Complete
85102T235030	85-102T23:50:30	Within specific year of any century
85102T2350	85-102T23:50	Ditto, with hour and minute only
102T2350	102T23:50	Ditto, within specific ordinal date in any year
102T23	102T23	Ditto, with hour only
85102T23	85-102T23	Within specific year of any century, with hour only
102T235030	102T23:50:30	Within specific ordinal date in any year of any century
etc.		
Combinations of calendar week, day number and local time of the day		
1985W155T235030	1985-W15-5T23:50:30	Complete
85W155T235030	85-W15-5T23:50:30	Within specific year of any century
85W155T2350	85-W15-5T23:50	Ditto, with hour and minute only
W155T2350	W15-5T23:50	Ditto, in any year
W155T23	W15-5T23	Ditto, with hour only
85W155T23	85-W15-5T23	Within specific year of any century, with hour only
W155T235030	W15-5T23:50:30	Within specific week and day of that week, in any century and year
etc.		
Combinations of day number and local time of the day		
5T235030	5T23:50:30	Any Friday, complete
5T2350	5T23:50	With hour and minute only
5T23	not applicable	With hour only

B.4 Periods of time

Basic format

Extended format

Period with specific start and specific end

A period beginning at 20 minutes and 50 seconds past 23 hours on 12 April 1985 and ending at 30 minutes past 10 hours on 25 June 1985

19850412T232050/19850625T103000

1985-04-12T23:20:50/1985-06-25T10:30:00

A period beginning on 12 April 1985 and ending on 25 June 1985

19850412/0625

1985-04-12/06-25

Duration of a period as a quantity of time

Two years, ten months, 15 days, 10 hours, 20 minutes and 30 seconds

P2Y10M15DT10H20M30S

not applicable

One year and six months

P1Y6M

not applicable

ISO 8601 : 1988 (E)

Seventy-two hours

PT72H

not applicable

Period with specific start and specific duration

A period of one year, 2 months, 15 days and 12 hours, beginning on 12 April 1985 at 20 minutes and 50 seconds past 23 hours

19850412T232050/P1Y2M15DT12H

1985-04-12T23:20:50/P1Y2M15DT12H

Period of specific duration and with specific end

A period of one year, 2 months, 15 days and 12 hours, ending on 12 April 1985 at 20 minutes and 50 seconds past 23 hours

P1Y2M15DT12H/19850412T232050

P1Y2M15DT12H/1985-04-12T23:20:50

ISO 8601 : 1988 (E)

UDC 529 : 003.62

Descriptors : information interchange, documentation, data representation, calendar dates.

Price based on 14 pages



INTERNATIONAL STANDARD ISO 8601 : 1988
TECHNICAL CORRIGENDUM 1

Published 1991-05-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION · МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ · ORGANISATION INTERNATIONALE DE NORMALISATION

Data elements and interchange formats — Information interchange — Representation of dates and times

TECHNICAL CORRIGENDUM 1

Éléments de données et formats d'échange — Échange d'information — Représentation de la date et de l'heure
RECTIFICATIF TECHNIQUE 1

Technical corrigendum 1 to International Standard ISO 8601 : 1988 was prepared by Technical Committee ISO/TC 154, *Documents and data elements in administration, commerce and industry*.

Page 6

Subclause 5.3

Lines 3 and 4, delete "[01] to [24]" and "[01] to [60]" and insert "[00] to [24]" and "[00] to [59]"

Subclause 5.3.1.3

Last line, delete "shall be preceded by a zero" and insert "shall be preceded by two zeros in accordance with 4.6"

Page 11

Annex B

Clause B.1, Calendar week — 15th week of 1985, extended format column, delete "-5W15" and insert "-5-W15"

SUPERSEDED

INTERNATIONAL STANDARD 2014

Reproduced By GLOBAL
ENGINEERING DOCUMENTS
With The Permission Of ISO
Under Royalty Agreement



INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION

Writing of calendar dates in all-numeric form

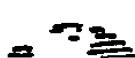
Représentation numérique des dates

First edition — 1976-04-01

UDC 529.2 : 003.35

Ref. No. ISO 2014-1976 (E)

Descriptors : calendar dates, writing, numeric representation.



Obtained From
GLOBAL ENGINEERING DOCUMENTS

Writing of calendar dates in all-numeric form

0 INTRODUCTION

In all forms of international traffic and exchange, dates must be clearly designated and able to be compared without any ambiguity.

This International Standard for writing of calendar dates in all-numeric form has been prepared to obviate the confusion arising from misinterpretation of the significance of the numerals in a date written with numerals only; it is considered that similar confusion does not arise when the month is spelled out, either in full or in abbreviated form.

The occasions on which an all-numeric date might be used have been examined and the advantages for these occasions of the descending order year—month—day have been found to outweigh those for the ascending order day—month—year, established in many parts of the world.

The advantages of this descending order include the following in particular :

- the ease with which the whole date may be treated as a single numeral for the purpose of filing and classification (for example for insurance or social security systems);
- arithmetic calculation, particularly in some computer uses;
- the possibility of continuing the order by adding digits for hour—minute—second.

1 SCOPE

This International Standard specifies the writing of dates of the Gregorian calendar in all-numeric form, signified by the elements year, month, day.

2 FIELD OF APPLICATION

This International Standard is applicable whenever a calendar date containing the elements year, month, day is written in all-numeric form.

3 RULES FOR WRITING CALENDAR DATES

3.1 Sequence

An all-numeric date shall be written in the following order :

year—month—day

3.2 Characters

An all-numeric date shall be expressed exclusively in arabic numerals, i.e. by using only the decimal digits 0, 1, 2, . . . , 9 and, if required, the hyphen (see 3.4).

3.3 Elements

An all-numeric date shall consist of

- four digits to represent the year;

NOTE — Two digits may be used where no possible confusion can arise from the omission of the century; however, four digits should be applied especially in correspondence and for documentation purposes to indicate clearly that the descending order is used.

- two digits to represent the month;
- two digits to represent the day.

3.4 Separator

Where a separator is used in an all-numeric date, only a hyphen or a space shall be used between year and month, and between month and day.

3.5 Examples

The 1st July 1976 shall be written in one of the following ways :

- a) 19760701
- b) 1976-07-01
- c) 1976 07 01

SUPERSEDED

INTERNATIONAL STANDARD 2014

Reproduced By GLOBAL
ENGINEERING DOCUMENTS
With The Permission Of ISO
Under Royalty Agreement



INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION

Writing of calendar dates in all-numeric form

Représentation numérique des dates

First edition — 1976-04-01

Ref. No. ISO 2014-1976 (E)

UDC 529.2 : 003.35

Descriptors : calendar dates, writing, numeric representation.

Writing of calendar dates in all-numeric form

0 INTRODUCTION

In all forms of international traffic and exchange, dates must be clearly designated and able to be compared without any ambiguity.

This International Standard for writing of calendar dates in all-numeric form has been prepared to obviate the confusion arising from misinterpretation of the significance of the numerals in a date written with numerals only; it is considered that similar confusion does not arise when the month is spelled out, either in full or in abbreviated form.

The occasions on which an all-numeric date might be used have been examined and the advantages for these occasions of the descending order year—month—day have been found to outweigh those for the ascending order day—month—year, established in many parts of the world.

The advantages of this descending order include the following in particular :

- the ease with which the whole date may be treated as a single numeral for the purpose of filing and classification (for example for insurance or social security systems);
- arithmetic calculation, particularly in some computer uses;
- the possibility of continuing the order by adding digits for hour—minute—second.

1 SCOPE

This International Standard specifies the writing of dates of the Gregorian calendar in all-numeric form, signified by the elements year, month, day.

2 FIELD OF APPLICATION

This International Standard is applicable whenever a calendar date containing the elements year, month, day is written in all-numeric form.

3 RULES FOR WRITING CALENDAR DATES

3.1 Sequence

An all-numeric date shall be written in the following order :

year—month—day

3.2 Characters

An all-numeric date shall be expressed exclusively in arabic numerals, i.e. by using only the decimal digits 0, 1, 2, . . . , 9 and, if required, the hyphen (see 3.4).

3.3 Elements

An all-numeric date shall consist of

- four digits to represent the year;

NOTE — Two digits may be used where no possible confusion can arise from the omission of the century; however, four digits should be applied especially in correspondence and for documentation purposes to indicate clearly that the descending order is used.

- two digits to represent the month;
- two digits to represent the day.

3.4 Separator

Where a separator is used in an all-numeric date, only a hyphen or a space shall be used between year and month, and between month and day.

3.5 Examples

The 1st July 1976 shall be written in one of the following ways :

- a) 19760701
- b) 1976-07-01
- c) 1976 07 01

March 1979

IEN: 85
RFC: 753

(8)

INTERNET MESSAGE PROTOCOL

Jonathan B. Postel

March 1979

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

(213) 822-1511

TABLE OF CONTENTS

PREFACE	iii
1. INTRODUCTION	1
1.1. Motivation	1
1.2. Scope	1
1.3. The Internetwork Environment	2
1.4. Operation	2
1.5. Interfaces	3
2. FUNCTIONAL DESCRIPTION	5
2.1. Relation to Other Protocols	5
2.2. Terminology	5
2.3. Assumptions	6
2.4. General Specification	7
2.5. Mechanisms	11
3. DETAILED SPECIFICATION	13
3.1. Overview of Message Structure	13
3.2. Data Elements	13
3.3. Message Objects	16
3.4. Command	23
3.5. Document	31
3.6. Message Structure	33
3.7. MPM Organization	36
3.8. Interfaces	39
4. EXAMPLES & SCENARIOS	41
Example 1: Message Format	41
Example 2: Delivery and Acknowledgment	43
GLOSSARY	49
REFERENCES	51
APPENDICES	53

PREFACE

This is the first edition of this specification and should be treated as a request for comments, advice, and suggestions. A great deal of prior work has been done on computer aided message systems and some of this is listed in the reference section. This specification was shaped by many discussions with members of the ARPA research community, and others interested in the development of computer aided message systems. This document was prepared as part of the ARPA sponsored Internetwork Concepts Research Project at ISI, with the assistance of Greg Finn, Alan Katz, Paul Mockapetris, and Mamie Chew.

Jon Postel

March 1979
IEN: 85
RFC: 753

J. Postel
USC-ISI
March 1979

INTERNET MESSAGE PROTOCOL

1. INTRODUCTION

This document describes an internetwork message system. The system is designed to transmit messages between message processing modules according to formats and procedures specified in this document. The message processing modules are processes in host computers. Message processing modules are located in different networks and together constitute an internetwork message delivery system.

This document is intended to provide all the information necessary to implement a compatible cooperating module of this internetwork message system.

1.1. Motivation

As computer supported message processing activities grow on individual host computers and in networks of computers, there is a natural desire to provide for the interconnection and interworking of such systems. This specification describes the formats and procedures of a general purpose internetwork message system, which can be used as a standard for the interconnection of individual message systems, or as a message system in its own right.

We also provide for the communication of data items beyond the scope of contemporary message systems. Messages can include typed segments which could represent drawings, or facsimile images, or digitized speech. One can imagine message stations equipped with speakers and microphones (or telephone hand sets) where the body of a message or a portion of it is recorded digitized speech. The output terminal could include a graphics display, and the message might present a drawing on the display, and verbally (via the speaker) describe certain features of the drawing. This specification provides basic data elements for the transmission of structured binary data, as well as providing for text transmission.

1.2. Scope

The Internet Message Protocol is intended to be used for the transmission of messages between networks. It may also be used for the local message system of a network or host. This specification was developed in the context of the ARPA work on the interconnection of networks, but it is anticipated that it has a more general scope.

The focus here is on the internal mechanisms to transmit messages, rather than the external interface to users. It is assumed that a number of user interface programs will exist. These will be both new programs designed to work with system and old programs designed to work with earlier systems.

1.3. The Internetwork Environment

The internetwork message environment consists of processes which run in hosts which are connected to networks which are interconnected by gateways. Each individual network consists of many different hosts. The networks are tied together through gateways. The gateways are essentially hosts on two (or more) networks and are not assumed to have much storage capacity or to "know" which hosts are on the networks to which they are attached [5].

1.4. Operation

The model of operation is that this protocol is implemented in a process. Such a process is called a Message Processing Module or MPM. The MPMs exchange messages by establishing full duplex communication and sending the messages in a fixed format described in this document. The MPM may also communicate other information by means of commands described here.

A message is formed by a user interacting with a User Interface Program or UIP. The user may utilize several commands to create various fields of the message and may invoke an editor program to correct or format some or all of the message. Once the user is satisfied with the messages it is "sent" by placing it in a data structure shared with the MPM.

The MPM discovers the unprocessed input data (either by a specific request or by a general background search), examines it, and using routing tables determines which outgoing link to use. The destination may be another user on this host, a user on another host in this network, or a user in another network.

In the first case, another user on this host, the MPM places the message in a data structure shared with the destination user, where that user's UIP will look for incoming messages.

In the second case, the user on another host in this network, the MPM transmits the message to the MPM on that host. That MPM then repeats the routing decision, and discovering the destination is local to it, places the messages in the data structure shared with the destination user.

In the third case, the user on a host in another network, the MPM transmits the messages to an MPM in that network if it knows how to establish a connection directly to it, otherwise the MPM transmits the message to an MPM that is "closer" to the destination. An MPM might not know of direct connections to MPMs in all other networks, but it must be able to select a next MPM to handle the message for each possible destination network.

A MPM might know a way to establish direct connections to each of a few MPMs in other nearby networks, and send all other messages to a particular big brother MPM that has a wider knowledge of the internet environment.

A individual network's message system may be quite different from the internet message system. In this case, intranet messages will be delivered using the network's own message system. If a message is addressed outside the network, it is given to a MPM which then sends it through the appropriate gateways via internet procedures and format to (or toward) the MPM in the destination network. Eventually, the message gets to a MPM on the network of the recipient of the message. The message is then sent via the local message system to that host.

When local message protocols are used, special conversion programs are required to transform local messages to internet format when they are going out, and to transform internet messages to local format when they come into the local environment. Such transformations are potentially information lossy. The internet message format attempts to provide features to capture all the information any local message system might use. However, a particular local message system is unlikely to have features equivalent to all the possible features of the internet message system. Thus, in some cases the transformation of an internet message to a local message discard of some of the information. For example, if an internet message carrying mixed text and speech data in the body is to be delivered in a local system which only carries text, the speech data may be replaced by the text string "There was some speech here". Such discarding of information is to be avoided when at all possible, and to be deferred as long as possible, still the possibility remains, that in some cases, it is the only reasonable thing to do.

1.5. Interfaces

The MPM calls on a reliable communication procedure to communicate with other MPMs. This is a Transport Level protocol such as the TCP [20]. The interface to such a procedure conventionally provides calls to open and close connections, send and receive data on a connection, and some means to signal and be notified of special conditions (i.e., interrupts).

March 1979

Internet Message Protocol
Introduction

The MPM receives input and produces output through data structures that are produced and consumed respectively by user interface (or other) programs.

2. FUNCTIONAL DESCRIPTION

2.1. Terminology

The basic unit transferred between networks is called a message. A message is made up of a transaction identifier (a number which uniquely identifies the message), a command list (which contains the necessary information for delivery), and the document list. The document list consists of a header and a body, which contains the actual data of the message.

For a personal letter the document body corresponds to the contents of a letter, the document header corresponds to the address and return address on the envelope.

For an inter-office memo the document body corresponds to the text, the document header corresponds to the header of the memo.

The commands correspond to the information used by the Post Office or the mail room to route the letter or memo.

The messages are routed by a process called the message processing module or MPM. Messages are created and consumed by User Interface Programs (UIPs) in conjunction with users.

Please see the Glossary section for a more complete list of terminology.

2.2. Assumptions

The following assumptions are made about the internetwork environment:

It is in general not known what format intranet addresses will assume. Since no standard addressing scheme would suit all networks, it is safe to assume there will be several and that they will change with time. Thus, frequent software modification throughout all internet MPMs would be required if such MPMs were to know about the formats on many networks. Therefore, each MPM which handles internet messages is required to know only the minimum necessary to deliver them.

We require each MPM to know completely only the addressing format of its own network. In addition, the MPM must be able to select an output link for each message addressed to another network or host. This does not preclude more intelligent behavior on the part of a given MPM, but at least this minimum is necessary. Each network has a unique name and number.

Each MPM will have a unique internet address. This feature will

March 1979

enable every MPM to place a unique "handling-stamp" on a message which passes through it en-route to delivery.

2.3. General Specification

There are several aspects to a distributed service to be specified. First there is the service to be provided, that is, the characteristics of the service as seen by its users. Second there is the service it uses, that is, the characteristics it assumes to be provided by some lower level service. And, third there is the protocol used between the modules of the distributed service.

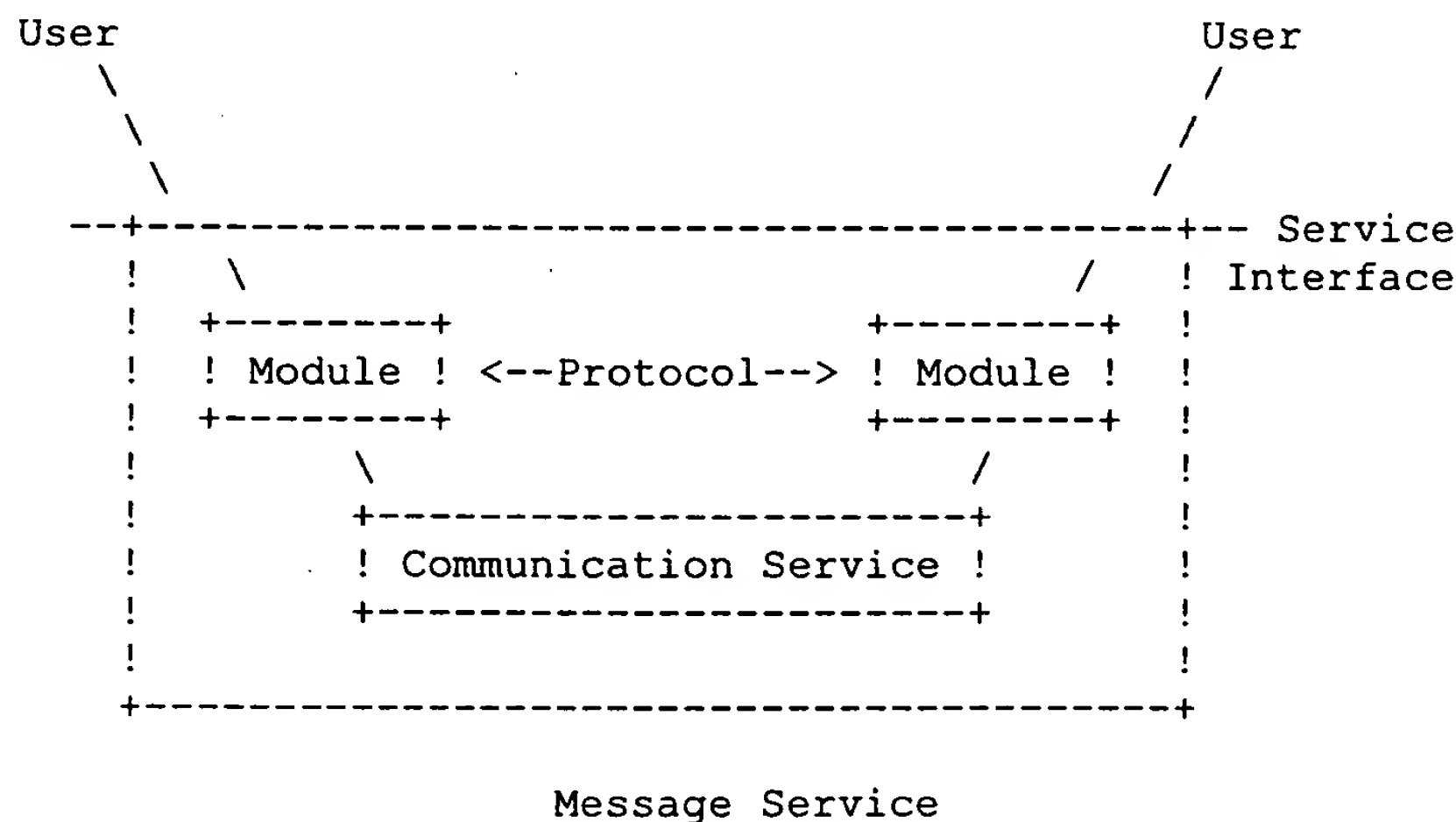


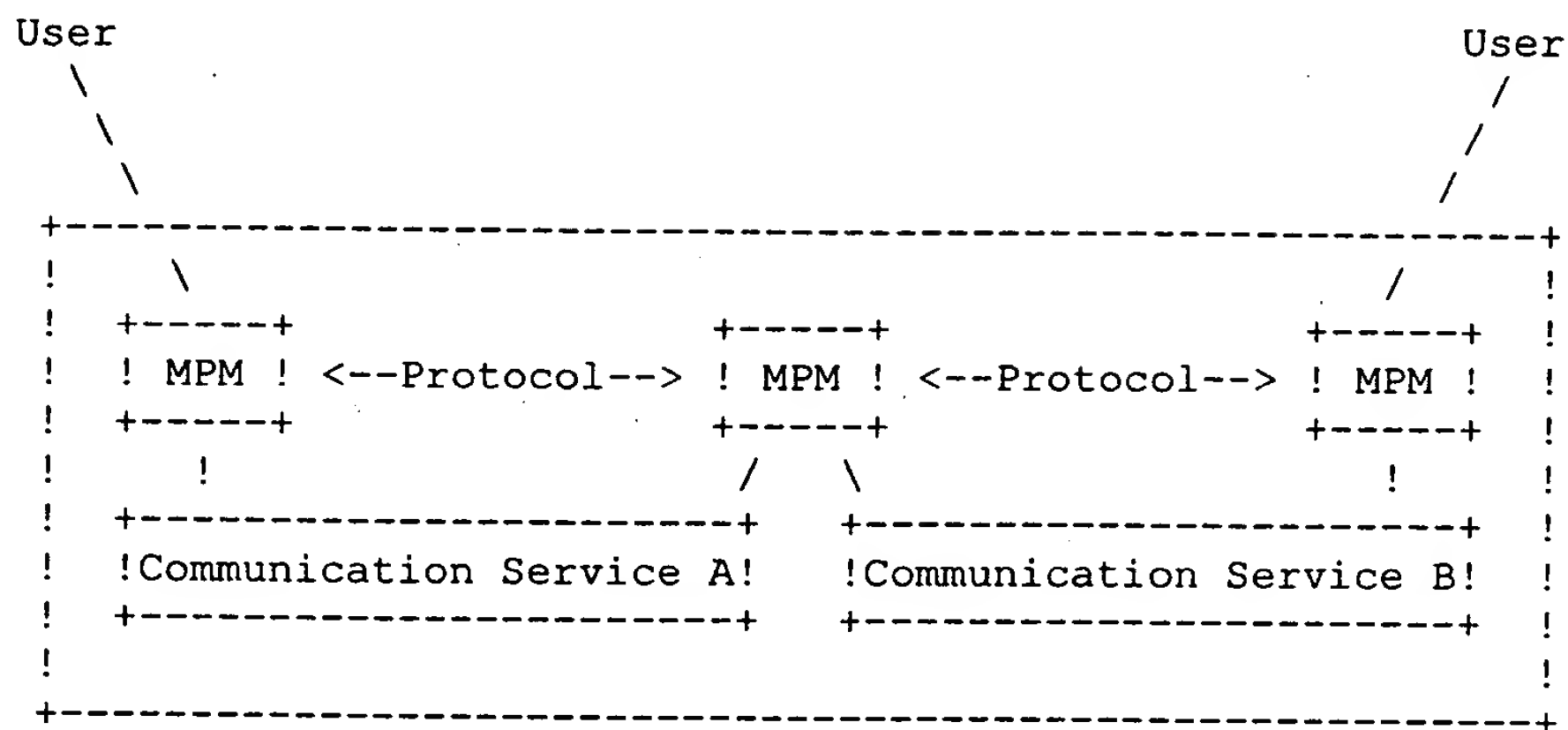
Figure 1.

The User/Message Service Interface

The service the message delivery system provides is to accept messages conforming to a specified format and to attempt to deliver those messages, and to report on the success or failure of the delivery attempt. This service is provided in the context of an interconnected system of networks, and may involve relaying a message through several intermediate MPMs utilizing different communication services.

The Message/Communication Service Interface

The message delivery system calls on a communication service to transfer information from one MPM to another. There may be different communication services used between different pairs of



Message Service with Internal Relaying

Figure 2.

The transfer of data between UIPs and MPMs is conceived of as the exchange of data structures which encode messages. The transfer of data between MPMs is also in terms of the transmission of structured data.

MPMs, though all communication services must meet the following service characteristics.

It is assumed that the communication service provides a reliable two way data stream. Such a data stream can usually be obtained in computer networks from the transport level protocol, for example, the Transmission Control Protocol (TCP) [20]. In any case the properties the communication service must provide are:

- o Logical connections for two way simultaneous data flow of arbitrary data (i.e., no forbidden codes). Data is delivered in the order sent with no gaps.
- o Simple commands to open and close the connections, and to send and receive data on the connections.
- o A way to signal and be notified "out-of-band" (such as TCP's urgent) is available so that some messages can be labeled "more important" than others.
- o Controlled flow of data so that data is not transmitted faster than the receiver chooses to consume it (on the average).
- o Transmission errors are corrected without user notification or involvement. Complete breakdown on communication is reported to the user.

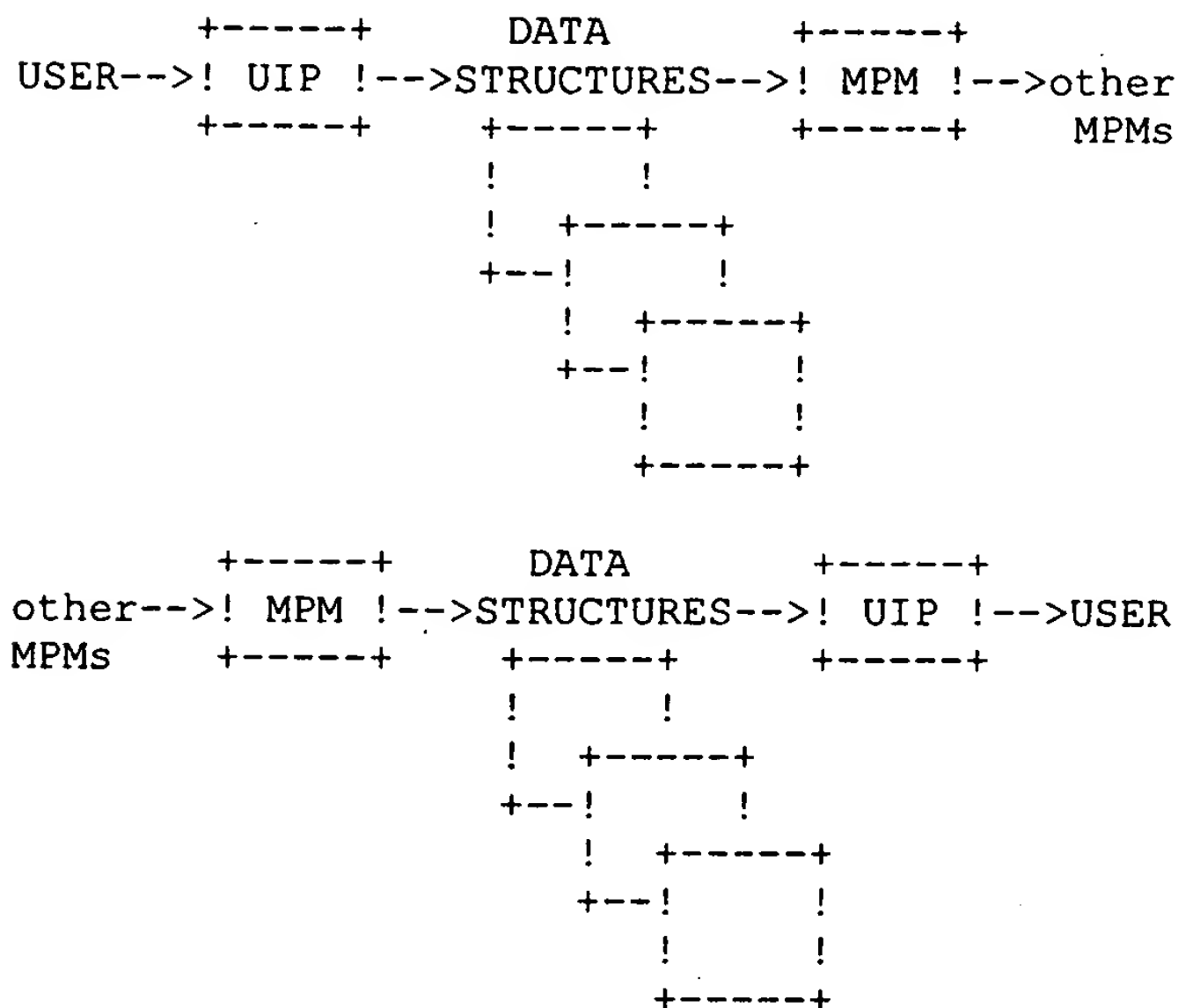
The Message-Message Protocol

The protocol used between the distributed modules of the message delivery system, that is, the MPMs is a small set of commands which convey requests and replies. These commands are encoded in a highly structured and rigidly specified format.

2.4. Mechanisms

MPMs are processes which use some communication service. A pair of MPMs which can communicate reside in a common interprocess communication environment. A MPM might exist in two (or more) interprocess communication environments, and such an MPM might act to relay messages between MPMs in the environments.

Internet Message Protocol
Functional Description

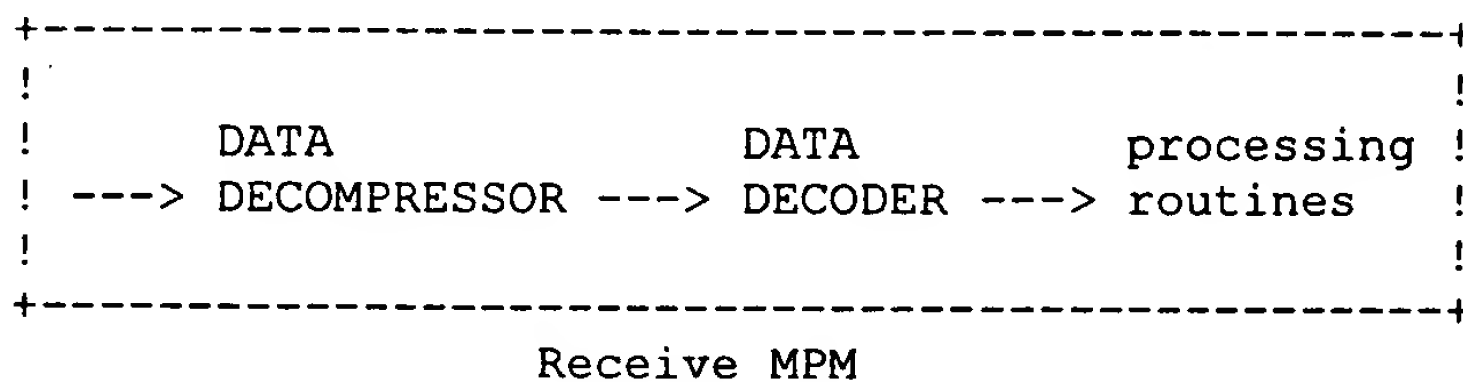
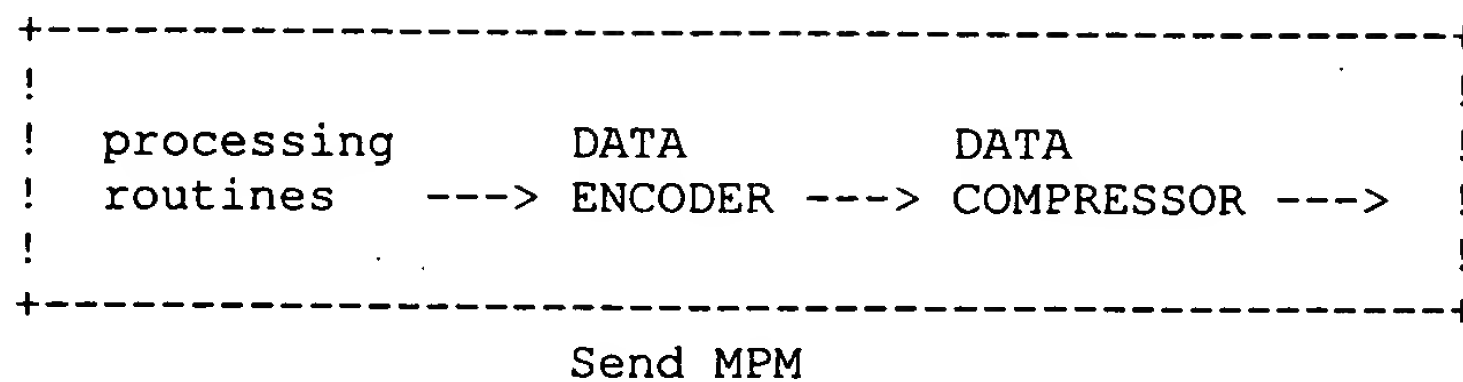


Message Flow

Figure 3.

In the following, a message will be described as a structured data object represented in a particular kind of typed data elements. This is how a message is presented when transmitted between MPMs or exchanged between an MPM and a UIP. Internal to a MPM (or a UIP), a message may be represented in any convenient form. As the following figure shows, when a message is ready for transmission, it moves from the processing routines to be encoded in the typed data elements and then to a data compression routine, and is finally transmitted. On the receiving side, the message is first decompressed then decoded from the data element representation to the local representation for the processing routines.

March 1979

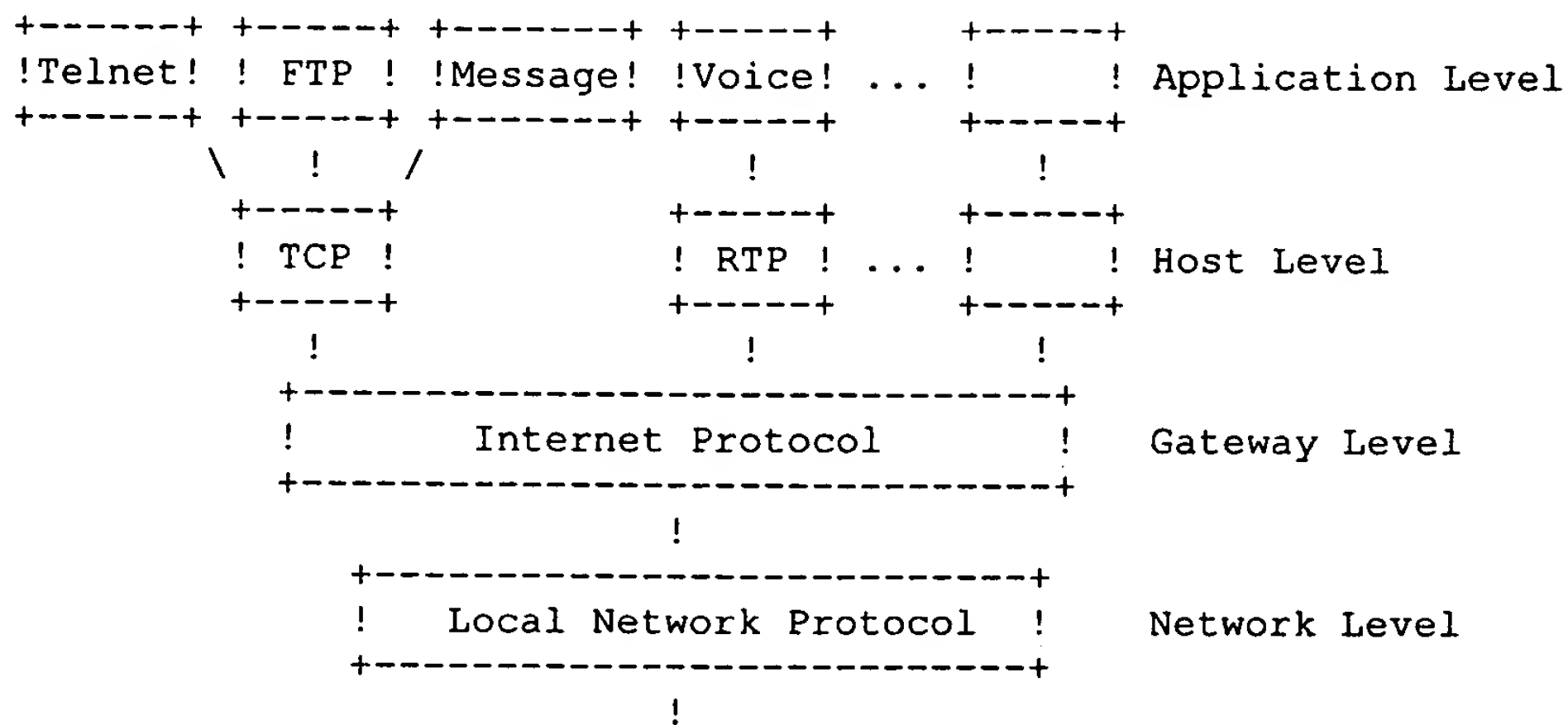


Detailed View

Figure 4.

2.5. Relation to Other Protocols

The following diagram illustrates the place of the message protocol in the protocol hierarchy:



Protocol Relationships

Figure 5.

The message protocol interfaces on one side to user interface programs and on the other side to a reliable transport protocol such as TCP.

3. DETAILED SPECIFICATION

The presentation of the information in this section is difficult since everything depends on everything, and since this is a linear media it has to come in some order. In this attempt, a very brief overview of the message structure is given, then a radical switch is made to defining the basic building blocks, and finally using the building blocks to reach the overall structure again.

3.1. Overview of Message Structure

In general a message is composed of three parts: the identification, the command, and the document. Each part is in turn composed of message objects.

The identification part is composed of a transaction number assigned by the originating MPM, and the internet host number of that MPM.

The command part is composed of an operation type, an operation code, an argument list, an error list, the destination mailbox, and a stamp. The stamp is a list of the MPMs that have handled this message.

The document part is composed of a header and a body. The message delivery system does not depend on the contents of the document part, but this specification does make some recommendations for the document header.

The following sections define the representation of a message as a structured object composed of other objects. Objects in turn are represented using a set of basic data elements.

3.2. Data Elements

The data elements defined here are similar to the data structure and encoding used in NSW [18].

Each of the diagrams which follow represent a sequence of octets. Field boundaries are denoted by the "!" character, octet boundaries by the "+" character. The diagrams are presented in left to right order. Each element begins with a one octet code.

Code ----	Type ----	Representation -----
0	No Operation	<pre> +-----+ ! 1 ! +-----+ </pre>
1	Padding	<pre> +-----+-----+-----+-----+-----+ ! 0 ! octet count ! Data ... +-----+-----+-----+-----+-----+ </pre>
2	Boolean	<pre> +-----+-----+ ! 2 ! 1/0 ! +-----+-----+ </pre>
3	Index	<pre> +-----+-----+-----+ ! 3 ! Data ! +-----+-----+-----+ </pre>
4	Integer	<pre> +-----+-----+-----+-----+-----+ ! 4 ! Data ! +-----+-----+-----+-----+-----+ </pre>
5	Bit String	<pre> +-----+-----+-----+-----+-----+ ! 5 ! bit count ! Data ... +-----+-----+-----+-----+-----+ </pre>
6	Text String	<pre> +-----+-----+-----+-----+-----+ ! 6 ! octet count ! Data ... +-----+-----+-----+-----+-----+ </pre>
7	List	<pre> +-----+-----+-----+-----+-----+-----+ ! 7 ! octet count ! item count ! Data +-----+-----+-----+-----+-----+-----+ </pre>
8	Proplist	<pre> +-----+-----+-----+-----+-----+ ! 8 ! octet count ! Data ... +-----+-----+-----+-----+-----+ </pre>

Element code 0 (NOP) is an empty data element used for padding when it is necessary. It is ignored.

Element code 1 (PAD) is used to transmit large amounts of data with a message for test or padding purposes. No action is taken with this data but the count of dummy octets must be correct to indicate the next element code.

Element code 2 (BOOLEAN) is a boolean data element which has the value 1 for True and 0 for False.

Element code 3 (INDEX) is a 16-bit unsigned integer datum. Element code 3 occupies only 3 octets.

Element code 4 (INTEGER) is a signed 32-bit integer datum. This will always occupy five octets. Representation is two's complement.

Element code 5 (BITSTR) is a bit string element for binary data. The bit string is padded on the right with zeros to fill out the last octet if the bit string does not end on an octet boundary. This data type must have the bit-count in the two octet count field instead of the number of octets.

Element code 6 (TEXT) is used for the representation of text. Seven bit ASCII characters are used, right justified in the octet. The high order bit in the octet is zero.

Element code 7 (LIST) can be used to create structures composed of other elements. The item-count contains the number of elements which follow. Any element may be used including List itself. The octet count specifies the number of octets in the whole list. A null or empty List, one with no elements, has an item-count of zero (0).

Element code 8 (PROPLIST) is the Property-List element. It has the following form:

```

+-----+-----+-----+-----+-----+
!   8   !   octet   ! pair !
!       !           count ! count!
+-----+-----+-----+-----+
                                +-----+-----+-----+-----+
                                ! name !   value   ! name   ! value   !
repeated ! count!   count   !     ...!     ...!
                                +-----+-----+-----+-----+

```

The Property-List structure consists of a set of unordered name/value pairs. The pairs are a one octet name count and a two octet value count followed by the name and value strings. The counts specify the length in octets of the name and value strings. Each string has a length in octets which agrees with its respective count. The count of octets until the next pair in the property list is 1 + 2 + name count + value count octets. The entire Property-List is of course equal in length to the octet count of the element itself. Immediately following the octet count for the entire element is a one octet pair count field which contains the total number of name/value pairs in the Proplist.

3.3. Message Objects

In the composition of messages we use a set of objects such as address, or date. These objects are encoded in the basic data elements. The message objects are built of data elements.

While data elements are typed, message objects are not. This is because messages are structured to the extent that only one kind of message object may occur in any position of a message structure.

The following is a list of some of the objects used in messages. The object descriptions are grouped by the section of the message in which they normally occur.

Identification

Internet Host Number (ihn)

This identifies a host in the internetwork environment. When used as a part of tid, it identifies the originating host of a message. The ihn is a 32 bit number, the higher order 8 bits identify the network, and the lower order 24 bits identify the host on that network.

INTEGER

Transaction Identifier (tid)

This is the transaction identifier associated with a particular command. It is a list of the transaction number and the internet host number of the originating host.

LIST (tn , ihn)

Transaction Number (tn)

This is a number which is uniquely associated with this transaction by the originating host. It identifies the transaction. (A transaction is a message and acknowledgment, this is discussed in more detail in later sections.) A tn must be unique for the time which the message (a request or reply) containing it could be active in the network.

INDEX

Command

Address

This is very similar to Mailbox in that it also is the "address" of a user. However, Address is intended to contain the minimum information necessary for delivery, and no more.

PROPLIST (---)

Answer

A yes (true) or no (false) answer to a question.

BOOLEAN

Arguments

This is the argument to many of the operations. It consists of a List of different data types. The List will have form and data relevant with the particular operation.

LIST (---)

Command-Type

Gives the type of a command (e.g., request, reply, alarm).

INDEX

Error-List

The error list contains information concerning an error which has occurred. It is a List comprised of the two objects error-class and error-string.

LIST (error class, error string)

Error-Class

A code for the class of the error.

INDEX

Error-String

A text string explaining the error.

TEXT

How-Delivered

A comment on the delivery of a messages, for instance a message could be delivered, forwarded, or turned over to general delivery.

LIST (TEXT)

Mailbox

This is the "address" of a user of the internetwork mail system. Mailbox contains information such as net, host, location, and local user-id of the recipient of the message. Some information contained in Mailbox may not be necessary for delivery.

As an example, when one sends a message to someone for the first time, he may include many items which are not necessary simply to insure delivery. However, once he gets a reply to this message, the reply could contain an Address (as opposed to Mailbox) which the user will use from then on.

A mailbox is a PROPLIST. A mailbox might contain the following name-value pairs:

name	element	description
----	-----	-----
IA	INTEGER	internet address
NET	TEXT	network name
HOST	TEXT	host name
USER	TEXT	user name
CITY	TEXT	city
COUNTRY	TEXT	country
STATE	TEXT	state
ZIP	TEXT	zip code
PHONE	TEXT	phone number

PROPLIST (---)

Operation

This names the operation or procedure to be performed.

TEXT

Options

REGULAR for normal delivery, FORWARD for message forwarding, GENDEL for general delivery, or other options which may be defined later.

LIST (TEXT, ...)

March 1979

Reasons

These could be mailbox does not exist, mailbox full, etc.

LIST (TEXT)

Stamp

Each MPM that handles the message must add a unique identifier (ihn, see above) to the list. This will prevent messages from being sent back and forth through the internet mail system without eventually either being delivered or returned to the sender.

LIST (ihn, ihn, ...)

Trail

When a message is sent through the internetwork environment, it acquires a list of MPMs that have handled the message in "Stamp". This list is then carried as "Trail" upon reply or acknowledgment of that message. More simply, requests and replies always have a "Stamp" and each MPM adds its ihn to this "Stamp." Replies, in addition, have a "Trail" which is the complete "Stamp" of the original message.

LIST (ihn, ihn, ...)

Type

The command type, e.g., request or reply.

INDEX

Document

In this section, we define some objects useful in message document headers. The ones we use are taken from the current ARPANET message syntax standard [6,8].

CC

When copies of a message are sent to others in addition to the addresses in the To object, those to whom the copies are sent will have their addresses recorded here. CC will be a single TEXT element.

TEXT

Date

The date and time are represented according to the International Standards Organization (ISO) recommendations [13,14,15]. Taken together the ISO recommendations 2014, 3307, and 4031 result in the following representation of the date and time:

yyyy-mm-dd-hh:mm:ss,fff+hh:mm

Where yyyy is the 4 digit year, mm is the two digit month, dd is the two digit day, hh is the two digit hour in 24 hour time, mm is the two digit minute, ss is the two digit second, and fff is the decimal fraction of the second. To this basic date and time is appended the offset from Greenwich as plus or minus hh hours and mm minutes.

TEXT

Document-Body

The document body will contain that portion of the message commonly thought of as the text portion. It will be composed of a list of elements. This will allow transmission of data other than pure text if such capabilities are needed. We can, for instance, envision digital voice communication through the transmission of BITSTR element, or transmission of graphic data, etc. Information regarding control of such features could be included in the header for cooperating sites, or in the body itself but such protocols would depend upon agreement among those sites involved. It is expected of course that the majority of messages will contain body portions comprised of TEXT elements.

LIST (---)

Document-Header

The document header contains the memo header presented to the user. In principle this may be of any style or structure. In this specification it is recommended that a PROPLIST be used and that the name-value pairs correspond to the header fields of RFC 733 [6].

PROPLIST (---)

From

The From is meant to be the name of the author of a document. It will be one TEXT element.

TEXT

Reply-To

Sometimes it will be desired to direct the replies of a message to some address other than the From or the Sender. In such a case the Reply-To object can be used.

TEXT

Sender

The Sender will contain the address of the individual who sent the message. In some cases this is NOT the same as the author of the message. Under such a condition, the author should be specified in the From object. The Sender is a single TEXT element.

TEXT

Subject

The subject of the message.

TEXT

To

To identifies the addressees of the message. The To object is one TEXT element.

TEXT

3.4. Command

This section describes the commands which processes in the internet message system can use to communicate. Several aspects of the command structure are based on the NSW Transaction Protocol [19]. The commands come in pairs, with each request having a corresponding reply.

A command is a list:

LIST (mailbox, stamp, type, operation, arguments, error-list)

The arguments are described generally here and more specifically, if necessary, in the description of each command.

mailbox: PROPLIST

This is the "to" specification of the message. Mailbox takes the form of a property list of general information, some of which is the essential information for delivery, and some of which could be extra information which may be helpful for delivery. Mailbox is different from address in that address is a very specific list without extra information.

stamp: LIST (INTEGER, ...)

This is a list of the MPMs that have handled the message. Each MPM must add its 32 bit Internet Host Number (ihn) to the LIST.

type: INDEX

type=1 a REQUEST operation.

type=2 a REPLY operation.

type=3 an ALARM operation. (A high priority message.)

type=4 a RESPONSE to an alarm operation.

operation: TEXT

Operation is the name of the operation or procedure to be performed. This string must be interpreted in an upper/lower case independent manner.

arguments: LIST

This is a list of arguments to the above operation.

error-list: LIST

If message is type 1 or 3 (a request or an alarm):

LIST () (a zero length list)

If message is a type 2 or 4 (a response or response to alarm)

LIST (error-class, error-string) indicates what, if any, error occurred

error-class: INDEX

=0: indicates success, no error

=1: partial results returned.

This error class is used when several steps are performed by one operation and some of them fail.

=2: failure, resources unavailable.

=3: failure, user error.

=4: failure, MPM error. Recoverable.

=5: failure, MPM error. Fatal.

=6: User abort requested

error-string: TEXT

This is a human readable character string describing the error.

Possible errors:

error-string	error-class
No errors	0
Command not implemented	2
Syntax error, command unrecognized	3
Syntax error, in arguments	3
Server error, try again later	4
No service available	5
User requested abort	6

command: DELIVER

type: 1

function: Sends message to a mailbox

reply: The reply is ACKNOWLEDGE

arguments: LIST (options)

options: one or more of the following

"REGULAR" regular delivery

"FORWARD" message forwarding

"GENDEL" general delivery

other options which may be defined later

argument structure:

LIST (LIST (TEXT, ...))

command: ACKNOWLEDGE

type: 2

function: reply to DELIVER

arguments: LIST (tid, trail, answer, reasons, how-delivered)

tid: tid of the originating message

trail: the stamp from the deliver command

answer: yes if delivered successfully,
no if error in delivery.

reasons: if the answer is yes, the reason is "ok", if the answer
is no the reason could be one of "no such user", "no such host",
"no such network", "address ambiguous", or a similar response

how-delivered: one or more of the following:

"FORWARD" message was accepted for forwarding

"GENDEL" message was accepted for general delivery

"ACCEPT" message was accepted for normal delivery

other types of delivery may be defined later

argument structure:

```
LIST ( LIST ( INDEX, INTEGER ),  
        LIST ( INTEGER, ... ),  
        BOOLEAN,  
        LIST ( TEXT ),  
        LIST ( TEXT ))
```

March 1979

Internet Message Protocol
Specification

command: PROBE

type: 1

function: finds out if specified mailbox (specified in mailbox of
the command) exists at a host

reply: the reply is RESPONSE

arguments: LIST (--none--)

argument structure:

LIST ()

March 1979

command: RESPONSE

type: 2

function: reply to PROBE

arguments: LIST (tid, trail, answer, address OR reasons)

tid: the tid which came from the originating PROBE

trail: the stamp which came from the originating PROBE

answer: Yes if mailbox found, or no for invalid mailbox

if answer is yes the fourth argument is address

if answer is no it is reasons

address: a specific address in the network

reasons: a reason why mailbox is invalid

Possible reasons include:

"Mailbox doesn't exist"

"Mailbox full"

"Mailbox has moved, try this new location", address

address is a new address to try

argument structure:

if answer is yes

```
LIST ( LIST ( INDEX, INTEGER ),  
        LIST ( INTEGER, ... ),  
        BOOLEAN,  
        PROPLIST )
```

if answer is no

```
LIST ( LIST ( INDEX, INTEGER ),  
        LIST ( INTEGER, ... ),  
        BOOLEAN,  
        LIST ( TEXT ) )
```

March 1979

Internet Message Protocol
Specification

command: CANCEL

type: 3

function: abort request for specified transaction

reply: The reply is CANCELED

arguments: LIST (tid)

tid of transaction to be cancelled

argument structure:

LIST (LIST (INDEX, INTEGER))

command: CANCELED

type: 4

function: reply to CANCEL

arguments: LIST (tid, trail, answer)

tid: tid of transaction to be cancelled

trail: the stamp of the CANCEL command

answer: yes if the command was canceled, no if not.

argument structure:

```
LIST ( LIST ( INDEX, INTEGER ),  
        LIST ( INTEGER, ... ),  
        BOOLEAN )
```

To summarize again, a command consists of a LIST of the following objects:

name	element
----	-----
mailbox	PROPLIST
stamp	LIST (INTEGER, ...)
type	INDEX
operation	TEXT
arguments	LIST (---)
error	LIST (INDEX, TEXT)

3.5. Document

The actual document follows the command list. It contains a header which usually contains such information as From, To, Date, CC, etc.; and the actual body of the message. The message delivery system does not depend on the document. The following section should be taken as a recommendation for common practice, not as a requirement.

Document Header

For the same reason that it is impossible to for see the many forms that intranet addresses will take, standardizing of document headers would also be a mistake. The approach we suggest is to lay the groundwork for a set of basic document header functions and provide for enough extensibility to allow nets to add whatever header features they desire. Features added in this fashion, however, may not be understood by other networks. It is suggested that subset defined here be implemented by all networks.

This subset is taken from the current ARPANET standard for message headers in the text oriented computer message system [6,8].

The document header will precede the document body portion of the message and will consist of a proplist data element. The document header is meant to be used by individual networks to tailor the header to suit their individual needs. As an example, consider the ARPA network. Typically, the receiver's name is taken to be his network address. It often prints in the document header in just that form: Frank@SITEX. Such a salutation is unacceptable in some more formal modes of communication. Some network might choose to place into header proplist the name-value pair ("SALUTATION:", "Mr. Frank Hacker"). Upon receipt of the message, the document handling program would then be able to scan the header proplist looking for such a pair and so be able to correctly address the recipient by name instead of by network address. However, other networks or

Internet Message Protocol
Specification

sites within the network may not understand such specific information. Under such a condition it should be ignored.

The minimum header is a PROPLIST of the following name-value pairs:

Name	Value
----	-----
DATE	TEXT
FROM	TEXT

A normal header is a PROPLIST containing the following name-value pairs:

Name	Value
----	-----
DATE	TEXT
SENDER	TEXT
FROM	TEXT
TO	TEXT
CC	TEXT
SUBJECT	TEXT

Document Body

The Body of the message is just a sequence of data elements which contains the actual document. Much of the time this will be a single TEXT element, but for some applications other data elements may be utilized.

LIST (---)

3.6. Message Structure

An internet message is composed of three parts. The first is the tid which identifies the transaction; the second is the Command List; and the third part is the Document List, which is itself comprised of a Document-Header and a Document-Body.

When shipped between two MPMs, a message will take the form of a LIST:

Message is:

```
LIST ( tid, Command-List, Document-List )
```

It is convenient to batch several messages together shipping them as a unit from one MPM to another. Such a group of messages is called a message-bag.

A message-bag will be a LIST of Messages, each Message is of the form described above.

Thus, a message-bag is:

```
LIST ( Message1, Message2, ... )
```

Message Sharing

When messages are batched for delivery, it may often be the case that the same Document will be sent to more than one recipient. Since the Document portion can usually be expected to be the major parts of the message, much repeated data would be sent if a copy of the Mail for each recipient were to be shipped in the message-bag.

To avoid this redundancy, messages are assembled in the message-bag so that actual data appears first and references to it appear later in the message-bag. Since each message has a unique tid, the references will indicate the tid of the actual data. In this sense, all references to copied data may be thought of as pointing earlier in the message-bag. The data to be retrieved can be thought of as indexed by tid. Note that the semantics require such references to point to data already seen.

When a portion is Shared, that portion is determined by its position within a message, i.e., if the Command list was to be Shared, then its position within a Message would contain the tid of the message already seen whose Command list was identical to it. The same is true of the Document Header and the Document Body. Only a complete Command, Header, or Body may be Shared, never a partial one.

If an encryption scheme is used, that portion of the message which is encrypted can not be shared. This is due to the fact that encrypting keys will be specific between two individuals.

Internal Message Organization

The tid

This is the transaction identifier. It is assigned by the originating MPM.

The Command List

The command-list is a LIST which contains two elements, content and command.

Content is one item of element type INDEX. If content=0, the item is not shared and the next element of the LIST is the command. If content=1 the item is shared. In this case, the second element will contain the tid of the command to share from. The tid must be of a prior message in the current message-bag. Other values of content may be defined later for different data structures.

Thus, command-list is:

LIST (content, tid) if content=1

Or,

LIST (content, command) if content=0

content is:

INDEX which is 0 if there is no sharing
 and is 1 if sharing occurs

tid is:

the tid of the message to be shared from

command is:

LIST (mailbox, stamp, type, operation, arguments, error-list)

The document-list

The document portion of an internet message is optional and when present is comprised of a LIST containing two elements:

document-list is:

LIST (header-list, body-list)

While either the header-list or the body-list may be shared, both elements must appear in the m.

The document-header

The header-list will be a List which will always contain two elements. The first element will be content to indicate whether or not the header is to be shared. The second element will either be the tid of the header to be copied (if content=1) or it will be the document-header (which is a PROPLIST) containing the actual header information (if content=0). The tid must point to a document-header already seen in the message-bag.

The header-list is either:

LIST (content, tid) if content=1

Or,

LIST (content, document-header) if content=0

document-header is:

PROPLIST which contains header information

The document-body

The body-list will be a LIST of two elements. The first element will again be content, indicating whether or not the body is to be shared. If it is shared, the second element will be tid indicating which body to copy. This tid must be of a message already seen in the message-bag. If content indicates no sharing, then the second item is a document-body.

body-list is:

LIST (content, tid) if content=1

Or,

LIST (content, document-body) if content=0

document-body is:

LIST (items comprising the body ...)

Message Fields

message := (tid, command-list, document-list)

tid := (tn, ihn)

command-list := (content, command)

command := (mailbox, stamp, type, operation,
arguments, error-list)

document-list := (header-list, body-list)

header-list := (content, document-header)

body-list := (content, document-body)

3.7. MPM Organization

Introduction

The heart of the internet message system is the MPM which is responsible for routing and delivering message between the networks. Each network must have at least one MPM. These MPMs are connected together, and internet mail is always transferred along channels between them. The system interfaces with the already existent local message system.

Since the local network message system may be very different from the internet system, special programs may be necessary to convert incoming internet messages to the local format. Likewise, messages outgoing to other networks may be converted to the internet format.

The MPM

Messages in the internet mail system are shipped in "bags," each bag containing one or more messages. Each bag is addressed to a specific MPM and contains messages for the hosts on that MPM's network.

Each MPM is expected to implement functions which will allow it to deliver local messages it receives and to forward non-local ones to other MPMs presumably closer to the message's destination.

Loosely, each MPM can be separated into five components:

1--Acceptor

Receives incoming Message-Bags, from other MPMs, from UIPs, or from conversion programs.

2--Message-Bag Processor

Splits a Bag into these three portions:

- a. Local Host Messages
- b. Local Net Messages
- c. Foreign Net Messages

3--Local Net Delivery

Delivers local net and local host messages, may call on conversion program.

4--Foreign Net Router

Creation of new Message-Bags for forwarding to other MPMs, determines route.

5--Foreign Net Shipper

Activates foreign shipping channels and ships Message-Bag to foreign MPMs. Performs data compression while shipping bags.

All of these components can be thought of as independent. Of the five, the Acceptor, the Local-Net Delivery, and the Message-Bag Processor are fully self-contained and communicate with each other only through a queue, the Bag-Input Queue. The function of the Acceptor is to await incoming Message-Bags and to insert them into the Bag-Input Queue.

That queue is the input to the Message-Bag Processor component which will separate and deliver suitable portions of the Message-Bags it retrieves from the queue to one of three queues:

- a. Local-Host Queue
- b. Local-Net Queue
- c. Foreign Net Queue

When a MPM decides to forward a message to another MPM, it must add its own identification (i.e., its ihn) to the stamp field of the command. The stamp then becomes a record of the route the message

has taken. An MPM should examine the stamp field to see if the message is in a routing loop. Some commands require the return of the stamp as a trail in the matching reply command.

All of these queues have as elements complete Message-Bags (some of which may have been portions of the original Bag).

The Local-Host and Local-Net queues serve as input to the Local-Net Delivery process. This component is responsible for delivering messages to its local host and other hosts on its local net to which it is connected. It must be capable of handling whatever error conditions the local net might return, including the ability to retransmit. It may call on conversion program to reformat the messages into a form the local protocol will accept. This will probably involve such things as copying shared information.

The other two processes are more closely coupled. The Foreign Net Router takes its input Bags from the Foreign Net Queue. From the internal information it contains, it determines which one of the MPMs to which it is connected should receive the Bag.

It then places the Bag along with the routing information into the Shippable Mail Queue. The Foreign Net Shipper retrieves it from that queue and transmits it across a channel to the intended foreign MPM.

The Foreign Net Router should be capable of receiving external input to its routing information table. This may come from the Foreign Net Shipper in the case of a channel going down, requiring a decision to either postpone delivery or to determine a new route.

The Router is responsible for maintaining sufficient topological information to determine where to forward any incoming Message-Bag. Decisions concerning the return of undeliverable Bags are made by the Router.

It should be stressed here that message delivery should be reliable. In the event that delivery is impossible, the message should be returned to the sender along with information regarding the reason for not delivering it.

Implementation Recommendations

Transaction numbers can be assigned sequentially with wrap around when the highest value is reached. This should ensure that no message with a particular transaction number from this source is in the network when another instance of this transaction number is chosen.

3.8. Interfaces

User Interface

It is assumed that the interface between the MPM and the UIP provides for passing data structures which represent the document portion of the message. In addition this interface must pass the delivery address information (which becomes the information in the mailbox field of the command). It is weakly assumed that the information is passed between the UIP and the MPM via shared files, but this is not the only possible mechanism. These two processes may be more strongly coupled (e.g., by sharing memory), or less strongly coupled (e.g., by communicating via logical channels).

Communication Interface

It is assumed here that the MPM use an underlying communication system, and TCP [20] has been taken as the model. Again, this is not intended to limit the implementation choices, other forms of interprocess communication are allowed and other types of physical interconnection are permitted. One might even use dial telephone calls to interconnect MPMs (using suitable protocols to provide reliable communication).

March 1979

Internet Message Protocol

4. EXAMPLES & SCENARIOS

Example 1: Message Format

Suppose we want to send the following message:

Date: 1979-03-29-11:46-08:00
From: Jon Postel <Postel@ISIB>
Subject: Meeting Thursday
To: Dave Crocker <DCrocker@Rand-Unix>
CC: Mamie

Dave:

Please mark your calendar for our meeting Thursday at 3 pm.

--jon.

It will be encoded in the structured format. The following will present successive steps in the top down generation of this message.

1. message
2. (tid, command-list, document-list)
3. ((tn, ihn),
 (content, command),
 (header-list, body-list))
4. ((tn, ihn),
 (content,
 (mailbox, stamp, type, operation,
 arguments, error-list)),
 ((content, document-header),
 (content, document-body)))
5. ((37, 167772404),
 (0, (
 (IA: 167772359, NET: arpa, HOST: rand-unix,
 USER: DCrocker),
 (167772404),
 1
 DELIVER
 ((REGULAR)),
 ())),
 ((0, (
 Date: 1979-03-29-11:46-08:00
 From: Jon Postel <Postel@ISIB>
 Subject: Meeting Thursday

March 1979

To: Dave Crocker <DCrocker@Rand-Unix>
CC: Mamie)),
(0, (Dave:

Please mark your calendar for our meeting
Thursday at 3 pm.

--jon.))))

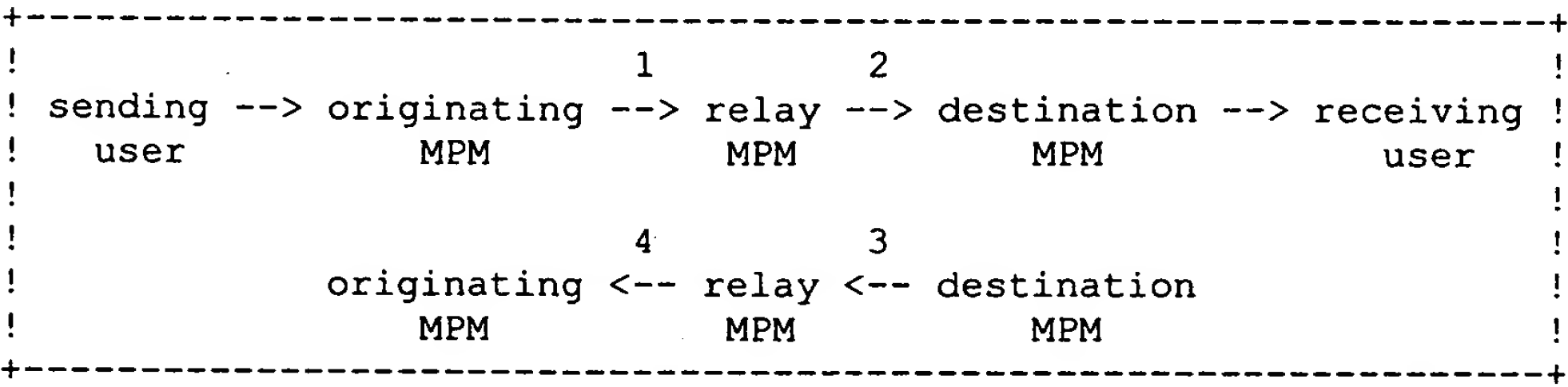
6. LIST(LIST(INDEX=37, INTEGER=167772404),
LIST(INDEX=0,
command LIST(PROPLIST(IA: 167772359,
NET: arpa,
mailbox HOST: rand-unix,
USER: DCrocker),
stamp LIST(INTEGER=167772404),
type INDEX=1
operation TEXT="DELIVER"
arguments LIST(LIST(TEXT="REGULAR")),
error-list LIST())),
LIST(LIST(INDEX=0,
document-header PROPLIST(
DATE: 1979-03-29-11:46-08:00
FROM: Jon Postel <Postel@ISIB>
SUBJECT: Meeting Thursday
TO: Dave Crocker <DCrocker@Rand-Unix>
CC: Mamie)),
LIST(INDEX=0,
document-body LIST(TEXT=
"Dave:

Please mark your calendar for
our meeting Thursday at 3 pm.

--jon."))))

Example 2: Delivery and Acknowledgment

The following is four views of the message of example 1 during the successive transmission from the origination MPM, through a relay MPM, to the destination MPM, and the return of the acknowledgment, through a relay MPM, to the originating MPM.



Transmission Path

Figure 6.

March 1979

1. Between the originating MPM and the relay MPM.

```
LIST( LIST( INDEX=37, INTEGER=167772404 ),
      LIST( INDEX=0,
command    LIST( PROPLIST( IA: 167772359,
                           NET: arpa,
mailbox    HOST: rand-unix,
                           USER: DCrocker ),
stamp      LIST( INTEGER=167772404 ),
type       INDEX=1
operation   TEXT="DELIVER"
arguments   LIST( LIST( TEXT="REGULAR" ) ),
error-list  LIST( ) ) ),
          LIST( LIST( INDEX=0,
document-header PROPLIST(
                  DATE: 1979-03-29-11:46-08:00
                  FROM: Jon Postel <Postel@ISIB>
                  SUBJECT: Meeting Thursday
                  TO: Dave Crocker <DCrocker@Rand-Unix>
                  CC: Mamie ) ),
document-body LIST( INDEX=0,
                  LIST( TEXT=
                      "Dave:

                      Please mark your calendar for
                      our meeting Thursday at 3 pm.

                      --jon." ) ) ) )
```

The originating MPM sends the message of example 1 to a relay MPM.

2. Between the relay MPM and the destination MPM.

```

LIST( LIST( INDEX=37, INTEGER=167772404 ),
      LIST( INDEX=0,
command    LIST( PROPLIST( IA: 167772359,
                        NET: arpa,
mailbox    HOST: rand-unix,
                        USER: DCrocker ),
stamp      LIST( INTEGER=167772404,
                        INTEGER=167772246 ),
type        INDEX=1
operation   TEXT="DELIVER"
arguments   LIST( LIST( TEXT="REGULAR" ) ),
error-list  LIST( ) ) ),
      LIST( LIST( INDEX=0,
document-header PROPLIST(
                        DATE: 1979-03-29-11:46-08:00
                        FROM: Jon Postel <Postel@ISIB>
                        SUBJECT: Meeting Thursday
                        TO: Dave Crocker <DCrocker@Rand-Unix>
                        CC: Mamie ) ),
document-body LIST( INDEX=0,
                        LIST( TEXT=
                                "Dave:

                                Please mark your calendar for
                                our meeting Thursday at 3 pm.

                                --jon." ) ) ) )

```

The relay MPM adds its ihn to the stamp, but otherwise the message is unchanged.

March 1979

3. Between the destination MPM and the relay MPM.

```
LIST( LIST( INDEX=1993, INTEGER=167772359 ),
      LIST( INDEX=0,
command    LIST( PROPLIST( IA: 167772404,
mailbox    USER: *MPM* ),
stamp      LIST( INTEGER=167772359 ),
type        INDEX=2
operation   TEXT="ACKNOWLEDGE"
arguments   LIST( LIST( INDEX=37,
tid          INTEGER=167772404 ),
              LIST( INTEGER=167772404,
trail         INTEGER=167772246,
              INTEGER=167772359 ),
answer      BOOLEAN=TRUE,
reason      LIST( TEXT="OK" ),
how-delivered LIST( TEXT="ACCEPT" ) ),
error-list  LIST( INDEX=0,
              TEXT="No Errors" ) ),
document    LIST( ) )
```

The destination MPM delivers the message to the user's UIP, and composes an acknowledgment. The acknowledgment is addressed to the originating MPM. Note that the trail is the stamp of the incoming message plus the ihn of the destination MPM.

4. Between the relay MPM and the originating MPM.

```

LIST( LIST( INDEX=1993, INTEGER=167772359 ),
      LIST( INDEX=0,
command    LIST( PROPLIST( IA: 167772404,
mailbox    USER: *MPM* ),
stamp      LIST( INTEGER=167772359
              INTEGER=167772246),
type        INDEX=2
operation    TEXT="ACKNOWLEDGE"
arguments    LIST( LIST( INDEX=37,
tid          INTEGER=167772404 ),
              LIST( INTEGER=167772404,
trail        INTEGER=167772246,
              INTEGER=167772359 ),
answer      BOOLEAN=TRUE,
reason      LIST( TEXT="OK" ),
how-delivered LIST( TEXT="ACCEPT" ) ),
error-list   LIST( INDEX=0,
document     TEXT="No Errors" ) ),
LIST( ) )

```

The relay MPM adds its ihn to the stamp and forwards the acknowledgment.

GLOSSARY

1822

BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

Command List

The part of a message used by the MPMs to determine the processing action to be taken.

datagram

A logical unit of data, in particular an internet datagram is the unit of data transferred between the internet module and a higher level module.

Destination

The destination address, an internet header datagram protocol field.

Document List

The part of the message created by or delivered to a user.

header

Control information at the beginning of a message, segment, datagram, packet or block of data.

IMP

The Interface Message Processor, the packet switch of the ARPANET.

Internet Address

A four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.

internet datagram

The unit of data exchanged between a pair of internet modules (includes the internet header).

Local Address

The address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.

Internet Message Protocol
Glossary

message

The unit of information transmitted between users of message systems. As transmitted between MPMs a message consists of a Transaction Identifier, a Command List, and a Document List.

module

An implementation, usually in software, of a protocol or other procedure.

MPM

A Message Processing Module, the process which implements this internet message protocol.

octet

An eight bit byte.

Rest

The 3 octet (24 bit) local address portion of an Internet Address.

RTP

Real Time Protocol: A host-to-host protocol for communication of time critical information.

Source

The source address, an internet header field.

TCP

Transmission Control Protocol: A host-to-host protocol for reliable communication in internetwork environments.

Transaction Identifier

The unique identifier of a message.

Type of Service

An internet datagram protocol header field which indicates the type (or quality) of service for this internet packet.

UIP

A User Interface Program, a program which presents message data to a user and accepts message data from a user. A program which interacts with the user in the composition and examination of messages.

XNET

A cross-net debugging protocol.

REFERENCES

- [1] Barber, D., and J. Laws, "A Basic Mail Scheme for EIN," INWG 192, February 1979.
- [2] Bhushan, A., K. Progran, R. Tomlinson, and J. White, "Standardizing Network Mail Headers," RFC 561, NIC 18516, 5 September 1973.
- [3] Bolt Beranek and Newman, "Specification for the Interconnection of a Host and an IMP," BBN Technical Report 1822, May 1978 (Revised).
- [4] Braaten, O., "Introduction to a Mail Protocol," Norwegian Computing Center, INWG 180, August 1978.
- [5] Cerf, V., "The Catenet Model for Internetworking," Information Processing Techniques Office, Defense Advanced Research Projects Agency, IEN 48, July 1978.
- [6] Crocker, D., J. Vittal, K. Progran, and D. Henderson, "Standard for the Format of ARPA Network Text Messages," RFC 733, NIC 41952, 21 November 1977.
- [7] Crocker, D., E. Szurkowski, and D. Farber, "Components of a Channel-independent Memo Transmission System," Department of Electrical Engineering, University of Delaware,, February 1979.
- [8] Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook," NIC 7104, for the Defense Communications Agency by the Network Information Center of SRI International, Menlo Park, California, Revised January 1978.
- [9] Harrenstien, K., "Field Addressing," ARPANET Message, SRI International, October 1977.
- [10] Haverty, J., "MSDTP -- Message Services Data Transmission Protocol," RFC 713, NIC 34739, April 1976.
- [11] Haverty, J., "Thoughts on Interactions in Distributed Services," RFC 722, NIC 36806, 16 September 1976.
- [12] Haverty, J., D. Henderson, and D. Oestreicher, "Proposed Specification of an Inter-site Message Protocol," 8 July 1975.
- [13] ISO-2014, "Writing of calendar dates in all-numeric form," Recommendation 2014, International Organization for Standardization, 1975.

March 1979

Internet Message Protocol
References

- [14] ISO-3307, "Information Interchange -- Representations of time of the day," Recommendation 3307, International Organization for Standardization, 1975.
- [15] ISO-4031, "Information Interchange -- Representation of local time differentials," Recommendation 4031, International Organization for Standardization, 1978.
- [16] Myer, T., and D. Henderson, "Message Transmission Protocol," RFC 680, NIC 32116, 30 April 1975.
- [17] Postel, J. "Internetwork Datagram Protocol, Version 4," USC Information Sciences Institute, IEN 80, February 1979.
- [18] Postel, J. "NSW Data Representation (NSWB8)," IEN 39, May 1978.
- [19] Postel, J. "NSW Transaction Protocol (NSWTP)," IEN 38, May 1978.
- [20] Postel, J. "Transmission Control Protocol, TCP, Version 4," USC Information Sciences Institute, IEN 81, February 1979.
- [21] Postel, J., "Assigned Numbers," RFC 750, NIC 45500, 26 September 1978.
- [22] Postel, J., "Message System Transition Plan," JBP 64, USC-Information Sciences Institute, February 1979.
- [23] Rivest, R. L. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" Communications of the ACM, Vol. 21, Number 2, February 1978.
- [24] Shoch, J., "A Note On Inter-Network Naming, Addressing, and Routing," Xerox Palo Alto Research Center, IEN 19, January 1978.
- [25] Thomas, R., "Providing Mail Services for NSW Users," BBN NSW Working Note 24, Bolt Beranek and Newman, October 1978.
- [26] White, J., "A Proposed Mail Protocol," RFC 524, NIC 17140, 13 June 1973.
- [27] White, J., "Description of a Multi-Host Journal," NIC 23144, 30 May 1974.
- [28] White, J., "Journal Subscription Service," NIC 23143, 28 May 1974.

APPENDICES

A. Encryption

It would be straightforward to add the capability to have the document portion of messages either wholly or partially encrypted. The approach is to define an additional basic data element to carry encrypted data. The data within this element could be composed of other elements, but that could only be perceived after the data was decrypted.

```

9  Encrypt      +-----+-----+-----+-----+-----+
                  !  9  !      octet count      ! Data ...
                  +-----+-----+-----+-----+-----+

```

Element code 9 (ENCRYPT) is Encrypt. The format is the one octet type code, the three octet type count, and count octets of data. Use of this element indicates that the data it contains is encrypted. The encryption scheme is yet to be decided but will probably be the Public Key Encryption technique [23] due to the capacity for coded signatures.

To process this, the user is asked for the appropriate key the first time an encryption block is seen for a particular message. The encrypted data is then decrypted. The data thus revealed will be in the form of complete data type fields. Encryption cannot occur over a partial field. The revealed data is then processed normally.

Note that there is no reason why all fields of a document could not be encrypted including all document header information such as From, Date, etc.

Internet Message Protocol Appendices

B. Data Compression

When message-bags are shipped between MPMs the data should be compressed according to the following scheme:

shipping-unit := compression-type message-bag

compression-type := A one octet compression type indicator.

compression-type value	description
-----	-----
0	no compression used
1	basic compression

basic compression

This basic compression procedure is the same as that defined for use with the ARPANET FTP [8]. Three types of compression-units may be formed, sequence-units, replication-units, and filler-units. The data is formed into a series of compression-units independent of the structure or object and element boundaries.

sequence-unit

A sequence-unit is a one octet flag and count followed by that many data octets.

```
+--+-----+-----+-----+-----+
!0!  n  !      n data octets ...
+--+-----+-----+-----+-----+
```

The flag and count octet has its high order bit zero and the remaining bits indicate the count (in the range 0 to 127) of following data octets.

replication-unit

A replication-unit is a one octet flag and count followed by one data octet, which is to be replicated count times.

```
+--+-----+-----+
!10!  n  !  data !
+--+-----+-----+
```

The flag and count octet has its high order two bits equal one-zero and the remaining six bits indicate the count (in the range 0 to 63) of number of time to replicate the data octet.

filler-unit

A filler-unit is a one octet flag and count, indicating that a filler octet is to be inserted count times.

```
+---+-----+
!11!  n  !
+---+-----+
```

The flag and count octet has its high order two bits equal one-one and the remaining six bits indicate the count (in the range 0 to 63) of number of time to insert the filler octet.

The filler octet is zero, the octet with all bits zero.

December 15, 1999



[Home](#) | [About Data Dimensions](#) | [Our Clients](#) | [Investor Relations](#) | [Publications](#) | [Contact Us](#)

e-Business

Healthcare

Outsourcing

**Software QA
and Testing**

**Enterprise
Application Integration**

**Enterprise
Knowledge Services**



Copyright © 1999, Data Dimensions, Inc. (DDI)

One Bellevue Center, Suite 2100

411 - 108th Avenue NE

Bellevue, WA 98004

All rights reserved.

Data Dimensions hereby authorizes you to view, print, copy, and distribute documents published by Data Dimensions on this World Wide Web server, provided that:

- a. The documents may be used (by you or any third party to whom you distribute them) only for informational, non-commercial purposes;
- b. No fee may be charged for distribution of any documents to any third party;
- c. Any and all copyright or other proprietary notices that appear on such documents, together with this permission notice, must appear on all copies that you make or distribute;
- d. You may not copy, host, or otherwise use any portion of the site in any world wide web location other than the one on which it is located.

Any product, process, or technology described in this document may be the subject of other Intellectual Property rights reserved by Data Dimensions and are not licensed hereunder.

Data Dimensions assumes no responsibility for the accuracy or completeness of the information contained in any document available through this World Wide Web server, and such information is subject to change without notice. DATA DIMENSIONS DISCLAIMS ALL WARRANTIES CONCERNING THIS INFORMATION, INCLUDING (WITHOUT LIMITATION) ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the United States Government is subject to restrictions set forth in DFARS 252.227-7013(c)(1)(ii) and FAR 52.227-19.

TRADEMARKS

Data Dimensions, the Data Dimensions logo, Ardes 2k, phrases containing Ardes 2k, Interactive Vendor Review, and Training Solutions 2000+ are registered trademarks, trademarks, or servicemarks of Data Dimensions, Inc. All other brand and product names mentioned herein are trademarks of their respective owners.

To benefit from Data Dimensions' expertise,
please call **877.DIAL.DDI (877.342.5334)**
or send e-mail to info@data-dimensions.com

Copyright

Headers for cached page www.data-dimensions.com/newsletters/journal/miljnl24.htm:
HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Date: Tue, 09 Nov 1999 04:12:36 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Fri, 22 Jan 1999 18:30:00 GMT
ETag: "0c44383546be1:150a"
Content-Length: 13760

The Millennium Journal

Vol. II.IV, July 1995

[Millennium Journal Index](#)

Date Storage Strategies

We are often confronted with questions about storage of date information. Many believe it is possible to reduce the millennium update work effort or minimize storage costs by working around the century. In this Journal we will explore some of the alternatives that we have employed and encountered.

We start this discussion with some reluctance. This stems from our belief that IT organizations should comply with industry and government standards. ANSI, FIPS, and ISO standards all state that the use of YY is permitted only when the date references the current century. SQL also defines date to include the century. For these reasons we have continued to insist that the only correct definition of year is CCYY.

Standards are created to minimize decision making. Where the CCYY standard is not followed, we see the amount of time required to complete the millennium update multiply. Costs often rise to \$1.50, and even \$3.00, per gross line of code.

Because it is necessary to review and determine how to handle every date calculation and comparison, we have found no significant savings in trying to retain 6-digit dates. Work-arounds cause production cycle times to increase. This requires additional work in performance tuning, and in acquiring new processor resources.

The number of storage options is infinite. We are providing here only a glimpse at the most common ones. The chaos that is caused in trying to keep track of what logic was used, documenting what the values mean, working around mistakes, and training new staff in all these rules costs the organization more in the long run. The variety should itself be a warning to managers to adopt a standard, and enforce it. Since most vendors are not following the standards in their response, the problem of passing date information will be complicated enough.

After all, this Millennium problem... is about time!

Richard Bergeon
Vice President, Technical Services

Three Approaches that are *NOT* Recommended

The following approaches to dealing with dates are not recommended. Imagine, if you can, an organization in which dates are presented in all the ways mentioned here. This could be your organization. Every format and technique that follows (and more) is being used by vendors as well as internal staff in order to "minimize" the amount of work involved in the update.

Single Digit Century Values

There are situations where space considerations and computational time requirements prohibit full century storage. In those situations, some turn to several single digit century options. A common solution is to store the century as a single digit logic flag.

This is the technique used by IBM's MVS operating system date. Old releases of MVS, CICS used a format 0CYYMMDD. This, and CYYMMDD, are the most common rule used when employing a single digit for the year. MVS 4.2 and CICS (EIBDATE) return values in the form of 0CYYDDD. In all these situations, C = "0" for "19" and "1" equals "20". But, even IBM is not consistent. In the AS/400 world, "0" is preceded the year in 1940 through 1999 and "1" is used for 2000 through 2039.

One of our clients began to use a one digit code for century: "0" = 1800, "1" = 1900, and "2" = 2000. Another variation we have seen is CYYMMDDS which uses signed values. The rule is that 1900 has the value of 0, 1 or +1 represents 2000, and -1 represents 1800. No doubt there are applications which use other numbers and even character values. Typical storage formats for single digit century are:

Field Description	Format	Field Length(1)
CYYMMDD	BINARY	2
CYYMMDDS	PACKED SIGNED	4
MMDDCYYS	PACKED SIGNED	4
CYYDDD	CHARACTER	6
MM/DD/CYY	CHARACTER	9
CYY/MM/DD	CHARACTER	9

(1) Field Lengths are given for IBM mainframes.
Other machines may have different lengths.

The use of the single-digit century does not save any update work, in fact, it adds to it since some logic must be installed to interpret the code and place the full century in the output fields.

Displaced Dates

Another technique we have seen used is the "displaced date". The year value stored is the difference calculated by subtracting the year from 100. The year 1995 is stored as the value "5". In 2005 the value stored will be "-5".

Some organizations use "9's compliment" - storing the difference the year subtracted from 99. An example, 95 subtract from 99 gives a value of 05. In 2005, 05 is subtracted from 99 giving 94. These techniques require the addition of an arithmetic step every place where the date is validated, stored, retrieved and displayed. Sorts can be tricky. The 9's compliment for the dates 98 through 02 are 02, 01, 99, 98, and 97.

One client adds 30 to the year, truncates the value to two positions, and then compares the value to 30. If the year is "95", the value is 25. Any value less than 30 is assigned a century value of 1900. If the year is "00" the calculated value becomes 30. The century is 2000. This arithmetic process is obviously associated with a logic process. It works a little better for sorts. Use of it becomes difficult if the field contains a date less than "70". It would necessary to use an increment of "40". Sometimes several increment values are used in the same program.

Logic-base Century Determination

Some organizations try desperately to hold on to their 6-digit date standard. Here are two examples of what is done to work around two-digit years. The first, "windowing", is only recommended as temporary measure.

In windowing, the two digit years are left alone in the files. A base year is selected (e.g., "50") where every year starting with (or, in some cases, greater than) "50" through "99" is treated as a 1900 date, and any year less than (or equal to) "50" is treated as 2000.

Sorts require an exit, and different date fields may require a different frame. Again, every place where dates are used in calculations, comparisons or displays, it is necessary to add additional logic. The biggest problem with using windowing is that it will allow all processes to continue to work... even while it produces incorrect results.

There is a logical way of determining the century that always works correctly. It is often possible to derive a logical answer about what century it is based on the contents of another field in the database. Here are two examples:

1. Calculation of current age (current year minus birth year) gives an answer of "09". This could mean 9 or 109 years of age. If a policy charge is high, then one can logically assume that the age is 109. A "relationship" indicator says the person is a "child", then the age is 9.
2. Calculation of mortgage expiration century can be determined by looking at the acquisition date for which the common routine has fixed the century - if the acquisition year is 1994, then the mortgage expiration date must be later (i.e., 44 must be 2044).

Lilian Dates

On October 15, 1582 the Gregorian Calendar was adopted. Its designer was Luigi Lilio - thus the origin of "Lilian". A Lilian clock is a counter incremented by the system at set intervals -

usually 100 ms. A subroutine calculates the date and time from the counter using a base date as the zero date.

While some products (e.g., IBM's LE/370) use the Gregorian origin date as the base date, it is not a standard. Use Lilian dates with caution.

1. Aligning Lilian clocks across different products can be tricky.

Common base dates are Jan. 1, 0000, Jan. 1, 1900, Jan. 1, 1964, and Jan. 4, 1980. The IBM MVS base date is 1/1/0000, DB/2's is 1/1/1800, ANSI COBOL reportedly is 1/1/1601, UNIX's clock is 01/01/1970, SAS starts at 01/01/1980, and VMS uses the Smithsonian date - November 17, 1858.

2. Know the limits of the system clock.

The counter field size determines the limit of the calendar - Macintosh's runs out on February 6, 2040. The C (computer language) calendar runs until January 18, 2038.

Satisfying the Standard

Meeting the standard may be possible without expanding data storage. Some organizations have saved space by adopting standard storage formats, eliminating storage formats that take more space.

Field Description	Format	Field Length(1)
YYYYMMDD	PACKED UNSIGNED (2)	4
YYYYDDDS	PACKED SIGNED	4
OYYYYMMDDS	PACKED SIGNED	5
YYYYDDD	CHARACTER	7
YYYYMMDD	CHARACTER	8

Some of our clients have elected to minimize the impact (on data entry procedures, report and screen formats, and century loading procedures) by retaining old formats and extending the year fields. The following formats meet standards, but are awkward to use in comparisons or calculations.

Field Description	Format	Field Length(1)
MMDDYYYY	PACKED UNSIGNED (2)	4
OMMDDYYYYS	PACKED SIGNED	5
MMDDYYYY	CHARACTER	8

(1) Field lengths are given for IBM mainframes. Other machines may have different word lengths.

(2) Packed unsigned number are not available under standard COBOL. User requires a call to a subroutine available in several date packages.

Data Dimensions' archives contain over forty different formats that organizations use to store dates. We have only mentioned a few here.

Our message: *If you want to simplify the work of the millennium update, stick with the standards. Trying to save time by adopting an alternative measure only costs more in the long run. There is no significant reduction in either analysis or testing time, and the dollars saved in storage are offset by the costs of additional processing time.*

Back to the [Millennium Journal Index](#)

July 1995, The Millennium Journal, Vol. II.IV
Copyright © 1995. All rights reserved. Data Dimensions, Inc.

The Millennium Journal

Vol. II.IV, July 1995

Date Storage Strategies

We are often confronted with questions about storage of date information. Many believe it is possible to reduce the millennium update work effort or minimize storage costs by working around the century. In this Journal we will explore some of the alternatives that we have employed and encountered.

We start this discussion with some reluctance. This stems from our belief that IT organizations should comply with industry and government standards. ANSI, FIPS, and ISO standards all state that the use of YY is permitted only when the date references the current century. SQL also defines date to include the century. For these reasons we have continued to insist that the only correct definition of year is CCYY.

Standards are created to minimize decision making. Where the CCYY standard is not followed, we see the amount of time required to complete the millennium update multiply. Costs often rise to \$1.50, and even \$3.00, per gross line of code.

Because it is necessary to review and determine how to handle every date calculation and comparison, we have found no significant savings in trying to retain 6-digit dates. Work-arounds cause production cycle times to increase. This requires additional work in performance tuning, and in acquiring new processor resources.

The number of storage options is infinite. We are providing here only a glimpse at the most common ones. The chaos that is caused in trying to keep track of what logic was used, documenting what the values mean, working around mistakes, and training new staff in all these rules costs the organization more in the long run. The variety should itself be a warning to managers to adopt a standard, and enforce it. Since most vendors are not following the standards in their response, the problem of passing date information will be complicated enough.

After all, this Millennium problem... is about time!

Richard Bergeon
Vice President, Technical Services

Three Approaches that are *NOT* Recommended

The following approaches to dealing with dates are not recommended. Imagine, if you can, an organization in which dates are presented in all the ways mentioned here. This could be your organization. Every format and technique that follows (and more) is being used by vendors as well as internal staff in order to "minimize" the amount of work involved in the update.

Single Digit Century Values

There are situations where space considerations and computational time requirements prohibit full century storage. In those situations, some turn to several single digit century options. A common solution is to store the century as a single digit logic flag.

This is the technique used by IBM's MVS operating system date. Old releases of MVS, CICS used a format 0CYYMMDD. This, and CYYMMDD, are the most common rule used when employing a single digit for the year. MVS 4.2 and CICS (EIBDATE) return values in the form of 0CYYDDD. In all these situations, C = "0" for "19" and "1" equals "20". But, even IBM is not consistent. In the AS/400 world, "0" is preceded the year in 1940 through 1999 and "1" is used for 2000 through 2039.

One of our clients began to use a one digit code for century: "0" = 1800, "1" = 1900, and "2" = 2000. Another variation we have seen is CYYMMDDS which uses signed values. The rule is that 1900 has the value of 0, 1 or +1 represents 2000, and -1 represents 1800. No doubt there are applications which use other numbers and even character values. Typical storage formats for single digit century are:

Field Description	Format	Field Length(1)
-----	-----	-----
CYYMMDD	BINARY	2
CYYMMDDS	PACKED SIGNED	4
MMDDCYYS	PACKED SIGNED	4
CYYDDD	CHARACTER	6
MM/DD/CYY	CHARACTER	9
CYY/MM/DD	CHARACTER	9
-----	-----	-----

(1) Field Lengths are given for IBM mainframes.
Other machines may have different lengths.

The use of the single-digit century does not save any update work, in fact, its adds to it since some logic must be installed to interpret the code and place the full century in the output fields.

Displaced Dates

Another technique we have seen used is the "displaced date". The year value stored is the difference calculated by subtracting the year from 100. The year 1995 is stored as the value "5". In 2005 the value stored will be "-5".

Some organizations use "9's compliment" - storing the difference the year subtracted from 99. An example, 95 subtract from 99 gives a value of 05. In 2005, 05 is subtracted from 99 giving 94. These techniques require the addition of an arithmetic step every place where the date is validated, stored, retrieved and displayed. Sorts can be tricky. The 9's compliment for the dates 98 through 02 are 02, 01, 99, 98, and 97.

One client adds 30 to the year, truncates the value to two positions, and then compares the value to 30. If the year is "95", the value is 25. Any value less than 30 is assigned a century value of 1900. If the year is "00" the calculated value becomes 30. The century is 2000. This arithmetic process is obviously associated with a logic process. It works a little better for sorts. Use of it becomes difficult if the field contains a date less than "70". It would necessary to use an increment of "40". Sometimes several increment values are used in the same program.

Logic-base Century Determination

Some organizations try desperately to hold on to their 6-digit date standard. Here are two examples of what is done to work around two-digit years. The first, "windowing", is only recommended as temporary measure.

In windowing, the two digit years are left alone in the files. A base year is selected (e.g., "50") where every year starting with (or, in some cases, greater than) "50" through "99" is treated as a 1900 date, and any year less than (or equal to) "50" is treated as 2000.

Sorts require an exit, and different date fields may require a different frame. Again, every place where dates are used in calculations, comparisons or displays, it is necessary to add additional logic. The biggest problem with using windowing is that it will allow all processes to continue to work... even while it produces incorrect results.

There is a logical way of determining the century that always works correctly. It is often possible to derive a logical answer about what century it is based on the contents of another field in the database. Here are two examples:

1. Calculation of current age (current year minus birth year) gives an answer of "09". This could mean 9 or 109 years of age. If a policy charge is high, then one can logically assume that the age is 109. A "relationship" indicator says the person is a "child", then the age is 9.
2. Calculation of mortgage expiration century can be determined by looking at the acquisition date for which the common routine has fixed the century - if the acquisition year is 1994, then the mortgage expiration date must be later (i.e., 44 must be 2044).

MMDDYYYY	PACKED UNSIGNED (2)	4
0MMDDYYYYS	PACKED SIGNED	5
MMDDYYYY	CHARACTER	8

-
- (1) Field lengths are given for IBM mainframes. Other machines may have different word lengths.
 - (2) Packed unsigned number are not available under standard COBOL. User requires a call to a subroutine available in several date packages.

Data Dimensions' archives contain over forty different formats that organizations use to store dates. We have only mentioned a few here.

Our message: If you want to simplify the work of the millennium update, stick with the standards. Trying to save time by adopting an alternative measure only costs more in the long run. There is no significant reduction in either analysis or testing time, and the dollars saved in storage are offset by the costs of additional processing time.

July 1995, The Millennium Journal, Vol. II.IV
Copyright © 1995. All rights reserved. Data Dimensions, Inc.

Intelligent Report Maintenance Using Dialogue Manager

"Best Copy Available"

*Gary Browe demonstrates
a date utility that simplifies
maintenance of FOCEXECs
that contain specific date
periods.*

Notice This Material May Be Protected
By Copyright Law (Title 17 U.S. Code)

by Gary Browe

Are your reports ready for the year 2000? Do they automatically change when the accounting period changes? Are they flexible and easily maintained? If not, then your FOCEXECs could benefit from the FOCUS date utility program discussed in this article.

Calculating the current accounting period is often necessary. For example, your customer needs a second quarter report, and it is currently the third quarter. This is a common situation in financial reporting, as reports must change to fulfill date requirements. To make accounting reports more automatic, create a FOCEXEC called DATES that contains the Dialogue Manager commands shown in Figure 1.

Figure 1 - DATES FOCEXEC

```

- * *****
- * Author: Gary C. Browe , Ford Motor Co.   File: DATES
- * Date: 12/89                               Release: 5.5.2
- * System: Any
- * Master(s): none
- * Function: Date calculations for FOCUS files.
- *
- * Remarks:  All &variable dates are in the form MMDDYYYY, using a
- *           4 digit year. This was done to avoid additional
- *           program logic to calculate dates between the 20th and
- *           21st century.
- *
- * Abbreviations: REF = reference   MTH = month   BEG = beginning
- *                CUR = current     YR  = year    END = ending
- *                ACT = accounting  PRE = previous
- * *****
- *
- * *****
- * The &variable REFDATE is set outside this FOCEXEC to change the
- * reference date to a date other than the current date. This is done
- * to fake FOCUS into "thinking" that it is some date other than today.
- * This is useful to "recreate" accounting reports for previous accounting
- * periods or preview future reports without changing program logic.
- * The reference date is typed in MMDDYY format and must refer to a date
- * from 01/01/1950 to 12/31/2049 for this focexec to work correctly.
- * *****
- *
- * IF &REFDATE.EXIST EQ 0 GOTO SETTODAY ELSE GOTO SETOTHER;
- * SETTODAY

```

(continued)

(Figure 1 - DATES FOCEXEC continued)

```

-SET &BASEDATE = &MDY;
-GOTO CONTINUE
-SETOTHER
-SET &BASEDATE = &REFDATE;
-CONTINUE
.*
.* *****
.* Current date settings.
.* Note: If the year is between 50 and 99, the date is assumed to be in
.* the 20th century (1900-1999) else the 21st century (2000-2099)
.* *****
-SET &HEADINGDT = EDIT (&BASEDATE, '99/99/99');
-SET &MTH = EDIT (&BASEDATE, '99$$$');
-SET &CURMTH = &MTH;
-SET &YR = EDIT (&BASEDATE, '$$$$99');
-SET &YR = IF &YR GE 50 AND &YR LE 99 THEN 19|&YR
ELSE 20|&YR;
-SET &CURYR = &YR;
-SET &BEGCURMTH = &MTH | 01 | &CURYR;
-SET &BEGCURYR = 0101 | &CURYR;
-SET &MIDCURYR = 0630 | &CURYR;
-SET &ENDCURYR = 1231 | &CURYR;
-SET &CURQTR = IF &MTH EQ 1 OR 2 OR 3 THEN 1
ELSE IF &MTH EQ 4 OR 5 OR 6 THEN 2
ELSE IF &MTH EQ 7 OR 8 OR 9 THEN 3
ELSE 4;
-SET &CURQTRNAME = IF &CURQTR EQ 1 THEN 'FIRST'
ELSE IF &CURQTR EQ 2 THEN 'SECOND'
ELSE IF &CURQTR EQ 3 THEN 'THIRD'
ELSE 'FOURTH';
.*
.* *****
.* Accounting Period: Determine the accounting period which is the
.* previous month or quarter.
.* *****
-SET &ACTQTR = IF &CURQTR - 1 EQ 0 THEN 4 ELSE &CURQTR - 1;
-SET &PREQTR = &ACTQTR;
-SET &ACTQTRNAME = IF &ACTQTR EQ 1 THEN 'FIRST'
ELSE IF &ACTQTR EQ 2 THEN 'SECOND'
ELSE IF &ACTQTR EQ 3 THEN 'THIRD'
ELSE 'FOURTH';
-SET &PREQTRNAME = &ACTQTRNAME;
-SET &MTH = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &ACTMTH = &MTH;
-SET &PREMTH = &ACTMTH;
-SET &YR = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &ACTYR = &YR;
-SET &BEGACTMTH = &MTH | 01 | &ACTYR;
-SET &MIDACTMTH = &MTH | 15 | &ACTYR;
-SET &BEGPREMTH = &BEGACTMTH;
-SET &MIDPREMTH = &MIDACTMTH;
-SET &BEGACTQTR = IF &ACTQTR EQ 1 THEN 0101 | &ACTYR
ELSE IF &ACTQTR EQ 2 THEN 0401 | &ACTYR
ELSE IF &ACTQTR EQ 3 THEN 0701 | &ACTYR
ELSE 0901 | &ACTYR;
-SET &BEGPREQTR = &BEGACTQTR;
-SET &ENDACTQTR = IF &ACTQTR EQ 1 THEN 0331 | &ACTYR
ELSE IF &ACTQTR EQ 2 THEN 0630 | &ACTYR
ELSE IF &ACTQTR EQ 3 THEN 0930 | &ACTYR
ELSE 1231 | &ACTYR;
-SET &ENDPREQTR = &ENDACTQTR;

```

(continued)

(Figure 1 - DATES FOCEXEC continued)

```

-SET &BEGACTYR      - 0101 | &ACTYR;
-SET &MIDACTYR      - 0630 | &ACTYR;
-SET &ENDACTYR      - 1231 | &ACTYR;
-SET &JAN           - 0101 | &ACTYR;
-SET &FEB           - 0201 | &ACTYR;
-SET &MAR           - 0301 | &ACTYR;
-SET &APR           - 0401 | &ACTYR;
-SET &MAY           - 0501 | &ACTYR;
-SET &JUN           - 0601 | &ACTYR;
-SET &JUL           - 0701 | &ACTYR;
-SET &AUG           - 0801 | &ACTYR;
-SET &SEP           - 0901 | &ACTYR;
-SET &OCT           - 1001 | &ACTYR;
-SET &NOV           - 1101 | &ACTYR;
-SET &DEC           - 1201 | &ACTYR;
* *****
* Previous year: is the accounting year - 1;
*
* Note: If the current month is January, the accounting year will be
*       the previous year AND the previous year will be the accounting
*       year - 1. This is done to allow previous year and accounting
*       year comparisons in Year End reports.
* *****
-SET &PREYR          - &ACTYR - 1;
-SET &BEGPREYR      - 0101 | &PREYR;
-SET &MIDPREYR      - 0630 | &PREYR;
-SET &ENDPREYR      - 1231 | &PREYR;
* *****
* Calculate two months ago: which is the current month - 2;
* *****
-SET &MTH            - IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH            - IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR             - IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &TWOMONTHAGO    - &MTH | 01 | &YR;
* *****
* Calculate three months ago: which is the current month - 3;
* *****
-SET &MTH            - IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH            - IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR             - IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &THREEMTHAGO    - &MTH | 01 | &YR;
* *****
* Calculate four months ago: which is the current month - 4;
* *****
-SET &MTH            - IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH            - IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR             - IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &FOURMTHAGO     - &MTH | 01 | &YR;
* *****
* Calculate five months ago: which is the current month - 5;
* *****
-SET &MTH            - IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH            - IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR             - IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &FIVEMTHAGO     - &MTH | 01 | &YR;
* *****
* Calculate six months ago: which is the current month - 6;
* *****
-SET &MTH            - IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH            - IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR             - IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &SIXMTHAGO      - &MTH | 01 | &YR;

```

These &variables, when resolved, will contain dates based on the system date or a reference date contained in the report request.

To understand how these &variables are resolved and what values they will contain after execution, execute the TESTDATE FOCEXEC shown in Figure 2 using the ECHO=ALL option (EX TESTDATE ECHO=ALL). The resolved &variables, shown in Figure 3, can then be used in screening statements, DEFINE or COMPUTE statements, column headings, or any other valid statements used in FOCUS.

Figure 2 - TESTDATE FOCEXEC

```

- * *****
- * Author: Gary C. Browe , Ford Motor Co.      File: TESTDATE
- * Date: 11/89                                Release: 5.5.2
- * System: Any
- * Master(s): none
- * Function: Test focexec to type the resolved &variables.
- * *****
- * REFDATE must be typed in MMDDYY format. If the reference date is
- * omitted, the system date will be substituted.
- * PROMPT &WHATDATE.16.TYPE THE REFERENCE DATE IN MMDDYY FORMAT.
- * SET &REFDATE = &WHATDATE.
- * INCLUDE DATES.
- * TYPE HEADINGDT - &HEADINGDT
- * TYPE CURMTH - &CURMTH
- * TYPE CURYR - &CURYR
- * TYPE BEGCURMTH - &BEGCURMTH
- * TYPE BEGCURYR - &BEGCURYR
- * TYPE MIDCURYR - &MIDCURYR
- * TYPE ENDCURYR - &ENDCURYR
- * TYPE CURQTR - &CURQTR
- * TYPE CURQTRNAME - &CURQTRNAME
- * TYPE ACTQTR - &ACTQTR
- * TYPE PREQTR - &PREQTR
- * TYPE ACTQTRNAME - &ACTQTRNAME
- * TYPE PREQTRNAME - &PREQTRNAME
- * TYPE ACTMTH - &ACTMTH
- * TYPE PREMTH - &PREMTH
- * TYPE ACTYR - &ACTYR
- * TYPE BEGACTMTH - &BEGACTMTH
- * TYPE MIDACTMTH - &MIDACTMTH
- * TYPE BEGPREMTH - &BEGPREMTH
- * TYPE MIDPREMTH - &MIDPREMTH
- * TYPE BEGACTQTR - &BEGACTQTR
- * TYPE BEGPREQTR - &BEGPREQTR
- * TYPE ENDACTQTR - &ENDACTQTR
- * TYPE ENDPREQTR - &ENDPREQTR
- * TYPE BEGACTYR - &BEGACTYR
- * TYPE MIDACTYR - &MIDACTYR
- * TYPE ENDACTYR - &ENDACTYR
- * TYPE JAN - &JAN
- * TYPE FEB - &FEB
- * TYPE MAR - &MAR
- * TYPE APR - &APR
- * TYPE MAY - &MAY
- * TYPE JUN - &JUN
- * TYPE JUL - &JUL
- * TYPE AUG - &AUG
- * TYPE SEP - &SEP
- * TYPE OCT - &OCT
- * TYPE NOV - &NOV
- * TYPE DEC - &DEC

```

(continued)

(Figure 2 - TESTDATE FOCEXEC continued)

-TYPE PREYR	-	&PREYR
-TYPE BEGPREYR	-	&BEGPREYR
-TYPE MIDPREYR	-	&MIDPREYR
-TYPE ENDPREYR	-	&ENDPREYR
-TYPE TWOMTHAGO	-	&TWOMTHAGO
-TYPE THREEMTHAGO	-	&THREEMTHAGO
-TYPE FOURMTHAGO	-	&FOURMTHAGO
-TYPE FIVEMTHAGO	-	&FIVEMTHAGO
-TYPE SIXMTHAGO	-	&SIXMTHAGO
-EXIT		

Figure 3 - Resolved &variables from TESTDATE FOCEXEC

HEADINGDT	-	01/01/90
CURMTH	-	01
CURYR	-	1990
BEGCURMTH	-	01011990
BEGCURYR	-	01011990
MIDCURYR	-	06301990
ENDCURYR	-	12311990
CURQTR	-	1
CURQTRNAME	-	FIRST
ACTQTR	-	4
PREQTR	-	4
ACTQTRNAME	-	FOURTH
PREQTRNAME	-	FOURTH
ACTMTH	-	12
PREMTH	-	12
ACTYR	-	1989
BEGACTMTH	-	12011989
MIDACTMTH	-	12151989
BEGPREMTH	-	12011989
MIDPREMTH	-	12151989
BEGACTQTR	-	09011989
BEGPREQTR	-	09011989
ENDACTQTR	-	12311989
ENDPREQTR	-	12311989
BEGACTYR	-	01011989
MIDACTYR	-	06301989
ENDACTYR	-	12311989
JAN	-	01011989
FEB	-	02011989
MAR	-	03011989
APR	-	04011989
MAY	-	05011989
JUN	-	06011989
JUL	-	07011989
AUG	-	08011989
SEP	-	09011989
OCT	-	10011989
NOV	-	11011989
DEC	-	12011989
PREYR	-	1988
BEGPREYR	-	01011988
MIDPREYR	-	06301988
ENDPREYR	-	12311988
TWOMTHAGO	-	11011989
THREEMTHAGO	-	10011989
FOURMTHAGO	-	09011989
FIVEMTHAGO	-	08011989
SIXMTHAGO	-	07011989

The DATES procedure is called into the report request via a -INCLUDE statement. The resolved &variables are then available for use (see Figure 4). For more complex reports or special conditions, other Dialogue Manager &variables can be defined using the DATES procedure as a base program (see Figure 5).

When using this technique, bear in mind that a limited number of Dialogue Manager variables are available. This technique makes extensive use of these variables; before using it, ensure that your existing application is not already at or near the limit.

In summary, using this Dialogue Manager DATES procedure throughout your database management system can reduce or eliminate FOCEXEC maintenance during major date changes. It will also have the added benefit of making your FOCEXECs more uniform, thus making them easier to interpret and maintain. ♦

Figure 4 - Sample FOCEXEC

```

*****
-* Author: Gary C. Browe , Ford Motor Co.   File: FIGURE4
-* Date: 12/89                               Release: 5.5.2
-* System: Any
-* Master(s): EMPLOYEE
-* Function: Sample example focexec using the DATES include file.
-* Remarks: The output from this focexec was based on the EMPLOYEE
            database supplied with PC/FOCUS release 3.1
*****
-SET &REFDATE = 080182 :
-INCLUDE DATES
*****
-* *****
-* Define XPAY_DATE as a FOCUS date.
-* *****
*****
DEFINE FILE EMPLOYEE
XPAY_DATE/MOYY = PAY_DATE;
END
*****
-* *****
-* Produce the monthly report from the employee database.
-* *****
TABLE FILE EMPLOYEE
HEADING CENTER
  "MONTHLY REPORT OF SALARY INCREASE"
  "FOR &ACTMTH / &ACTYR </1"
PRINT
  XPAY_DATE AS "PAY DATE"
  GROSS AS "GROSS PAY"
  BY LAST_NAME AS "LAST NAME"
  BY FIRST_NAME AS "FIRST NAME"
  IF XPAY_DATE GE "BEGACTMTH"
  IF XPAY_DATE LT "BEGCURMTH"
ON TABLE SUBFOOT
  " <45 "
  "TOTAL GROSS <42 <TOT.GROSS "
END
-EXIT

```

Output

08/01/82
MONTHLY REPORT OF SALARY INCREASE
FOR 07 / 1982

LAST NAME	FIRST NAME	PAY DATE	GROSS PAY
BLACKWOOD	ROSEMARIE	07/30/1982	\$1,815.00
CROSS	BARBARA	07/30/1982	\$2,255.00
IRVING	JOAN	07/30/1982	\$2,238.50
JONES	DIANE	07/30/1982	\$1,540.00
MCKNIGHT	ROGER	07/30/1982	\$1,342.00
ROMANS	ANTHONY	07/30/1982	\$1,760.00
SMITH	MARY	07/30/1982	\$1,100.00
STEVENS	ALFRED	07/30/1982	\$916.67
TOTAL GROSS			\$12,967.17

Figure 5 - Sample FOCEXEC

```

- *
*****
- * Author: Gary C. Browe , Ford Motor Co.      File: FIGURES
- * Date: 11/89                                Release: 5.5.2
- * System: Any
- * Master(s): EMPLOYEE
- * Function: Example focexec using dates file with additional dates
- *              calculated.
- * Remarks: The output from this focexec was based on the EMPLOYEE
- *              database supplied with PC/FOCUS release 3.1
- *

```

```

*****
-SET &REFDATE = 080183 ;
-INCLUDE DATES
-SET &2YRSAGO = &PREYR - 1;
-SET &BEG2YRSAGO = 0101 | &2YRSAGO;
-SET &END2YRSAGO = 1231 | &2YRSAGO;
- *
- * *****
- * Define XPAY_DATE as a FOCUS date.
- * *****
- *

```

```

DEFINE FILE EMPLOYEE
XPAY_DATE/MOY - PAY_DATE;
2YRSAGOPAY/D10.2M - IF XPAY_DATE GE '&BEG2YRSAGO'
AND XPAY_DATE LE '&END2YRSAGO' THEN GROSS ELSE 0;
PREYRPAY/D10.2M - IF XPAY_DATE GE '&BEGPREYR'
AND XPAY_DATE LE '&ENDPREYR' THEN GROSS ELSE 0;
ACTYRPAY/D10.2M - IF XPAY_DATE GE '&BEGACTYR'
AND XPAY_DATE LE '&ENDACTYR' THEN GROSS ELSE 0;
END
TABLE FILE EMPLOYEE
HEADING CENTER
"&HEADINGDT"
"EMPLOYEE PAY HISTORY REPORT </1>"
-SUM
2YRSAGOPAY AS &2YRSAGO GROSS EARNINGS
PREYRPAY AS &PREYR GROSS EARNINGS
ACTYRPAY AS &ACTYR GROSS EARNINGS
BY LAST_NAME AS LAST_NAME
BY FIRST_NAME AS FIRST_NAME
ON TABLE COLUMN-TOTAL
END
-EXIT

```



Gary Browe is a programmer analyst at Ford Motor Company, Parts and Service Division and president of the Detroit-Toledo FOCUS Users Group, TED McFUSE. His experience includes application design, end-user consulting, and giving FOCUS application presentations to FUSE groups. He has been using FOCUS since 1983.

Output

08/01/83
EMPLOYEE PAY HISTORY REPORT

LAST NAME	FIRST NAME	1981 GROSS EARNINGS	1982 GROSS EARNINGS	1983 GROSS EARNINGS
BLACKWOOD	ROSEMARIE	\$.00	\$1,815.00	\$.00
CROSS	BARBARA	\$.00	\$2,255.00	\$.00
IRVING	JOAN	\$.00	\$2,238.50	\$.00
JONES	DIANE	\$.00	\$1,540.00	\$.00
MCKNIGHT	ROGER	\$.00	\$1,342.00	\$.00
ROMANS	ANTHONY	\$.00	\$1,760.00	\$.00
SMITH	MARY	\$.00	\$1,100.00	\$.00
STEVENS	ALFRED	\$.00	\$916.67	\$.00
TOTAL		\$.00	\$12,967.17	\$.00

#17

SAS[®] Language: Reference

Version 6
First Edition



SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc., 1990. 1042 pp.

SAS® Language: Reference, Version 6, First Edition

Copyright © 1990 by SAS Institute Inc., Cary, NC, USA.

ISBN 1-55544-381-8

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

Restricted Rights Legend. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, March 1990

2nd printing, June 1992

3rd printing, June 1993

4th printing, July 1994

5th printing, September 1995

6th printing, December 1996

Note that text corrections may have been made at each printing.

The SAS® System is an integrated system of software providing complete control over data access, management, analysis, and presentation. Base SAS software is the foundation of the SAS System. Products within the SAS System include SAS/ACCESS®, SAS/AF®, SAS/ASSIST®, SAS/CALC®, SAS/CONNECT®, SAS/CPE®, SAS/DMI®, SAS/EIS®, SAS/ENGLISH®, SAS/ETS®, SAS/FSP®, SAS/GRAPH®, SAS/IMAGE®, SAS/IML®, SAS/IMS-DLI®, SAS/INSIGHT®, SAS/LAB®, SAS/NVISION®, SAS/OR®, SAS/PH-Clinical®, SAS/QC®, SAS/REPLAY-CICS®, SAS/SESSION®, SAS/SHARE®, SAS/SPECTRAVIEW®, SAS/STAT®, SAS/TOOLKIT®, SAS/TRADER®, SAS/TUTOR®, SAS/DB2®, SAS/GEO®, SAS/GIS®, SAS/PH-Kinetics®, SAS/SHARE*NET®, and SAS/SQL-DS™ software. Other SAS Institute products are SYSTEM 2000® Data Management Software, with basic SYSTEM 2000, CREATE®, Multi-User®, QueX®, Screen Writer®, and CICS interface software; InfoTap® software; NeoVisuals® software; JMP®, JMP IN®, and JMP Serve® software; SAS/RTERM® software; and the SAS/C® Compiler and the SAS/CX® Compiler; Video Reality™ software; VisualSpace™ software; Budget Vision®, CFO Vision®, Compensation Vision®, IT Service Vision®, and Risk Vision™ software; Scalable Performance Data Server™ software; and Emulus® software. MultiVendor Architecture™ and MVA™ are trademarks of SAS Institute Inc. SAS Institute also offers SAS Consulting®, and SAS Video Productions® services. *Authorline®*, *Books by Users™*, *The Encore Series®*, *JMP® Cable®*, *Observations®*, *SAS Communications®*, *SAS Professional Services™*, *SAS Views®*, the *SASware Ballot®*, *SelecText™*, and *Solutions@Work™* documentation are published by SAS Institute Inc. The SAS Video Productions logo, the Books by Users SAS Institute's Author Service logo, and The Encore Series logo are registered service marks or registered trademarks of SAS Institute Inc. The Helplus logo, the SelecText logo, the SAS Online Samples logo, the Video Reality logo, the Quality Partner logo, the SAS Business Solutions logo, and SAS Rapid Warehousing Program logo are service marks or trademarks of SAS Institute Inc. All trademarks above are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

The Institute is a private company devoted to the support and further development of its software and related services.

Other brand and product names are registered trademarks or trademarks of their respective companies.

DOC S18, Ver 1.66W,012290

Table 3.8 (continued)

Category	Informat	Description
	\$HEXw.	converts hexadecimal data to character data.
	\$OCTALw.	converts octal data to character data.
	\$PHEXw.	converts packed hexadecimal data to character data.
	\$VARYINGw.	reads varying-length character values.
	\$w.	reads standard character data.
Date and time informats	DATEw.	reads data values (ddmmmyy).
	DATETIMEw.	reads date and time values (ddmmmyy hh:mm:ss.ss).
	DDMMYYw.	reads day-month-year (ddmmyy).
	JULIANw.	reads Julian dates (yyddd or yyyyddd).
	MMDDYYw.	reads month-day-year (mmddy).
	MONYYw.	reads month and year (mmmyy).
	MSECw.	reads TIME MIC values.
	NENGOW.	reads Japanese dates values (r.yymmdd).
	PDTIMEw.	reads packed decimal time of SMF and RMF records.
	RMFDURw.	reads the duration of RMF measurement intervals in RMF records.
	RMFSTAMPw.	reads time and date fields of RMF records.
	SMFSTAMPw.	reads time-date values of SMF records.
	TIMEw.	reads hours, minutes, and seconds (hh:mm:ss.ss).
	TODSTAMPw.	reads 8-byte time-of-day stamp.
	TUw.	reads timer units.
	YYMMDDw.	reads year, month, and day (yymmdd).
	YYQw.	reads quarters of the year.
Column binary	\$CBw.	reads standard character data from column-binary files.
	CBw.d	reads standard numeric values from column-binary files.
	PUNCH.d	reads whether a row of column-binary data is punched.
	ROWw.d	reads a column-binary field down a card column.
Numeric	BINARYw.d	converts positive binary values to integers.
	BITSw.d	extracts bits.
	BZw.d	converts blanks to 0s.
	COMMAw.d	removes embedded commas, decimal points, and parentheses.

(continued)

Table 3.10 (continued)

Category	Format	Description
Date and Time Formats	DATEw.	writes data values (ddmmmyy).
	DATETIMEw.d	writes datetime values (ddmmmyy:hh:mm:ss.ss).
	DAYw.	writes day of month.
	DDMMYYw.	writes date values (ddmmyy).
	DOWNAMEw.	writes name of day of the week.
	HHMMw.d	writes hours and minutes.
	HOURw.d	writes hours and decimal fractions of hours.
	JULDAYw.	writes Julian day of the year.
	JULIANw.	writes Julian dates (yyddd or yyyyddd).
	MMDDYYw.	writes month-day-year (mmddy).
	MMSSw.d	writes minutes and seconds.
	MMYYxw.	writes month and year separated by a character. (MMYYxw. is a set of formats.)
	MONNAMEw.	writes name of month.
	MONTHw.	writes month of year.
	MONYYw.	writes month and year (mmmyy).
	NENGOW.	writes Japanese dates (r.yymmdd).
	QTRw.	writes quarter of year.
	QTRRw.	writes quarter of year in Roman numerals.
	TIMEw.d	writes hours, minutes, and seconds.
	TODw.	writes the time portion of datetime values.
	WEEKDATEw.	writes day of week and date (day-of-week, month-name dd yy).
	WEEKDATXw.	writes day of week and date (day-of-week, dd month-name yy).
	WEEKDAYw.	writes day of week.
	WORDDATEw.	writes date with name of month, day, and year (month-name dd, yyyy).
	WORDDATXw.	writes date with day, name of month, and year (dd month-name yyyy).
	YEARw.	writes year part of date value.
	YYMMxw.	writes year and month, separated by a character. (YYMMxw. is a set of formats.)
	YYMMDDw.	writes year-month-day (yymmdd).
	YYMONw.	writes year and month abbreviation.
	YYQxw.	writes year and quarter separated by a character. (YYQxw. is a set of formats.)
	YYQRxw.	writes year and quarter in Roman numerals separated by a character. (YYQRxw. is a set of formats.)

(continued)

contain characters that represent special missing values that allow you to distinguish between types of missing values.

For more information on reading numeric data, including details about missing values in input data and how the SAS System handles invalid data, see "Reading Raw Data" in Chapter 2.

Representing Numeric Data in SAS Programs

Numeric data can be represented in SAS programs as constants in standard numeric, scientific, and hexadecimal notation. Ordinary numeric missing values are represented as single periods. Special missing values can be represented as a period followed by one of the characters A through Z or the underscore character (_).

Bit masks can be used in bit testing to compare internal bits in a value's representation.

For more information on how to represent numeric data in SAS programs, including the details of bit testing, see "SAS Constants" in Chapter 4.

Writing Numeric Data

The *w.d* format is the default format for writing numeric data. However, there are numerous other SAS formats you can use in a *FORMAT*, an *ATTRIB*, or a *PUT* statement for writing numeric data.

If the value of a variable does not fit into the width of the format you are using, the SAS System tries to squeeze the value into the space available, in which case numeric formats may revert to the *BESTw.* format. If it is not possible to represent the value in some fashion, the SAS System prints asterisks. Even if the SAS System truncates or rounds a value of a numeric variable during the printing routine, the complete value is maintained and is stored intact by the system. (You can use the *PROBSIG=* system option to control the precision of statistical procedures.)

For information on individual formats, see Chapter 14. For information on the *PROBSIG=* system option, see Chapter 16.

Using SAS Date and Time Values

The SAS System represents date, time, and datetime values as numbers using the following rules:

- A date is represented by the number of days between January 1, 1960 and that date.
- A time is represented as the number of seconds between midnight and that time of day.
- A datetime is represented by the number of seconds between midnight, January 1, 1960 and that datetime.

The SAS System has a number of informats that read date and time values and convert them to SAS date and time values. The SAS System also has a number of formats that write date and time values in a variety of ways.

SAS dates are valid back to A.D. 1582 and ahead to A.D. 20,000. Leap year, century, and fourth-century adjustments are handled correctly. However, leap seconds are ignored, and the SAS System does not adjust for daylight saving time.

DATE system option

controls printing of date and time values. When this option is enabled, the SAS System prints on the top of each page of output the date and time the SAS job started. When you are running the SAS System in interactive mode, the time the job started is the time you started your SAS session.

CENTER system option

controls whether output is centered. By default, the SAS System centers titles and procedure output on the page and on the terminal display.

See Chapter 16 for more information about using the SAS system options discussed in this section.

Reformatting Values

Certain SAS statements, procedures, and options enable you to print values using specified formats. You can apply or change formats with the FORMAT and ATTRIB statements, or with the VAR window in display manager. The SAS formats available are described in Chapter 14, "SAS Formats." See Chapter 9 for complete descriptions of the FORMAT and ATTRIB statements and Chapter 17 for details on the VAR window.

The FORMAT procedure enables you to design your own formats and informats, giving you added flexibility in displaying values. See Chapter 18, "The FORMAT Procedure," in the *SAS Procedures Guide* for more information.

Printing Missing Values

The SAS System represents ordinary missing numeric values on SAS output as a single period, and missing character values as a blank space. If you have specified special missing values for numeric variables, the SAS System writes the letter or the underscore. For character variables, the SAS System writes a series of blanks equal to the length of the variable.

The MISSING= system option enables you to specify a character to print in place of the period for ordinary missing numeric values. See Chapter 16 for a discussion of the MISSING= system option.

Changing the Destination of the Log and Output

You can redirect both the SAS log and procedure output to your terminal display, to a printer, or to an external file. You can redirect output using the following methods:

PRINTTO procedure

routes DATA step, log, or procedure output from the system default destinations to the destination you choose. See Chapter 28 in the *SAS Procedures Guide* for more information.

DACCTAB

**Accumulated depreciation
from specified tables**

Financial

HELP FUNCTION

Syntax

DACCTAB(*p*,*v*,*t1*, . . . ,*tn*)

Description

- p* is numeric, the period for which the calculation is to be done. For noninteger *p* arguments, the depreciation is prorated between the two consecutive time periods preceding and following the fractional period.
- v* is numeric, the depreciable initial value of the asset.
- t1*, . . . ,*tn* are numeric, the fractions of depreciation for each time period.

The DACCTAB function returns the accumulated depreciation using user-specified tables, given by

$$\text{DACCTAB}(p, v, t_1, \dots, t_n) = \begin{cases} 0 & p \leq 0 \\ v(t_1 + t_2 + \dots + t_{\text{int}(p)} + (p - \text{int}(p))t_{\text{int}(p)+1}) & 0 < p < n \\ v & p \geq n \end{cases}$$

For a given *p*, only the arguments *t1*, . . . ,*tk* need to be specified with *k*=ceil(*p*).

Example

An asset has a depreciable initial value of \$1000 and a five-year lifetime. Using a table of the annual depreciation rates of .15, .22, .21, .24, and .20 during the first, second, third, fourth, and fifth years, respectively, the accumulated depreciation throughout the third year can be expressed as

```
y3=DACCTAB(3,1000,.15,.22,.21,.24,.20);
```

The value returned is 580.00. The fourth rate, .24, and the fifth rate, .20, can be omitted since they are not needed in the calculation.

DATE

**Returns the current date as
a SAS date value**

Date and time

HELP FUNCTION

Syntax

DATE()

Description

The DATE function produces the current date as a SAS date value representing the number of days between January 1, 1960 and the current date. The alias for the DATE function is TODAY.

Example

The following statements illustrate the DATE function:

```
tday=date();
if (tday-datedue)> 15 then
  do;
    put 'As of ' tday date7. 'Account #'
      account 'is more than 15 days overdue.';
  end;
```

See Also

TODAY function

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language.”

DATEJUL

Converts a Julian date to a SAS date value

Date and time

HELP FUNCTION

Syntax

DATEJUL(*julian-date*)

Description

The DATEJUL function converts a Julian date to a SAS date value.

You can use the following argument with the DATEJUL function:

<i>julian-date</i>	specifies any valid SAS numeric expression representing a Julian date in the form <i>yyddd</i> or <i>yyyyddd</i> , where <i>yy</i> or <i>yyyy</i> represents the year and <i>ddd</i> the day of the year. Two-digit year values are based on the YEARCUTOFF= system option. The value of <i>ddd</i> must be between 1 and 365 (or 366 for a leap year).
--------------------	---

Examples

The following statements return values of 10957 and 14976:

```
□ start=datejul(89365);
□ end=datejul(2001001);
```

See Also

JULDATE function

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language.”

DATEPART

Extracts the date from a SAS datetime value

Date and time

HELP FUNCTION

Syntax

DATEPART(*datetime*)

Description

The DATEPART function returns a SAS date value corresponding to the date portion of a SAS datetime value.

You can use the following argument with the DATEPART function:

datetime specifies a SAS expression representing a SAS datetime value.

Example

The following example of the DATEPART function returns a value equivalent to February 1, 1989. The PUT statement writes the resulting value as **February 1, 1989**:

```
/* CONN shows when the computer session began */
conn='01feb89:8:45'dt;
servdate=datepart(conn);
put servdate worddate.;
```

See Also

DATETIME and TIMEPART functions

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language.”

DATETIME

Returns the current date and time of day

Date and time

HELP FUNCTION

Syntax

DATETIME()

Description

The DATETIME function returns the current date and time of day as a SAS datetime value.

Example

The following example returns a SAS value representing the number of seconds between January 1, 1960 and the current date:

```
when=datetime();
put when=;
```

See Also

DATE and TIME functions

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language.”

DAY

Returns the day of the month from a SAS date value

Date and time

HELP FUNCTION

Syntax

DAY(*date*)

Description

The DAY function produces an integer representing the day of the month from a SAS date value. Integers can range from 1 through 31.

You can use the following argument with the DAY function:

date specifies a SAS expression representing a SAS date value.

Example

The following statements assign a value of 5 to D:

```
now='05may89'd;  
d=day(now);
```

See Also

MONTH and YEAR functions

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language.”

DEPDB

Declining balance depreciation

Financial

HELP FUNCTION

Syntax

DEPDB(*p,v,y,r*)

Description

p is numeric, the period for which the calculation is to be done. For noninteger *p* arguments, the depreciation is prorated between the two consecutive time periods preceding and following the fractional period.

v is numeric, the depreciable initial value of the asset.

y is numeric, the lifetime of the asset, with $y > 0$.

r is numeric, the rate of depreciation expressed as a fraction, with $r \geq 0$.

The DEPDB function returns the depreciation using the declining balance method, given by

$$\text{DEPDB}(p,v,y,r) = \text{DACCDB}(p,v,y,r) - \text{DACCDB}(p-1,v,y,r) \quad .$$

The *p* and *y* arguments must be expressed using the same units of time. A double declining balance is obtained by setting *r* equal to 2.

DATEw.**Reads date values
(ddmmmyy)**

Date and time

Width range: 7–32

Default width: 7

HELP INFORMAT

Description

The DATEw. informat reads date values in the form *ddmmmyy* or *ddmmmyyyy*, where *dd* is an integer from 01 through 31 representing the day of the month, *mmm* is the first three letters of the month name, and *yy* or *yyyy* is an integer representing the year. Blanks and other special characters can be placed between day, month, and year values. Width values must allow space for blanks and special characters.

Note: The SAS System defaults to a date in the 1900s if *yy* is two digits. Use the YEARCUTOFF= system option to override the system default and specify a date range of your choice.

Example

Data Lines	SAS Statement	Results
-----1-----2		
1jan1990	input day date10.;	10958
01 jan 90		10958
1 jan 90		10958
1-jan-1990		10958

See Also

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

DATE function in Chapter 11, “SAS Functions”

DATEw. format in Chapter 14, “SAS Formats”

YEARCUTOFF= system option in Chapter 16, “SAS System Options”

DATETIMEw.**Reads datetime values
(ddmmmyy hh:mm:ss.ss)**

Date and time

Width range: 13–40

Default width: 18

HELP INFORMAT

Description

The DATETIMEw. informat reads datetime values in the form *ddmmmyy*, followed by a blank or special character, and *hh:mm:ss.ss*. The value of *hh* ranges from 00 through 23 and the values of *mm* and *ss* range from 00 through 59. The value of *ss.ss* is an optional time value representing seconds and decimal fractions of seconds. With the DATETIMEw. informat you must give a value both for the date and the time.

Note: The SAS System defaults to a date in the 1900s if *yy* is two digits. Use the YEARCUTOFF= system option to override the system default and specify a date range of your choice.

Example

Data Lines	SAS Statement	Results
-----+-----1-----+-----2		
23dec89:10:03:17.2	input date datetime20.;	946029797.2
23dec1989/10:03:17.2		946029797.2

See Also

DATEw. and TIMEw. informats

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

DATETIME function in Chapter 11, “SAS Functions”

DATEw., DATETIMEw.d, and TIMEw.d formats in Chapter 14, “SAS Formats”

YEARCUTOFF= system option in Chapter 16, “SAS System Options”

DDMMYYw.

Reads date values (ddmmyy)

Date and time

Width range: 6–32

Default width: 6

HELP INFORMAT

Description

The DDMMYYw. informat reads date values in *ddmmyy* form, where *dd*, *mm*, and *yy* are integers representing the day, month, and year. The day, month, and year fields can be separated by blanks or special characters. However, if delimiters are used, they should be placed between all fields in the value. Blanks can also be placed before and after the date.

Note: The SAS System defaults to a date in the 1900s if *yy* is two digits. Use the YEARCUTOFF= system option to override the system default and specify a date range of your choice.

Example

Data Lines	SAS Statement	Results
-----+-----1-----+-----2		
231090	input day ddmmyy8.;	11253
23/10/90		11253
23 10 90		11253

See Also

DATEw., MMDDYYw., and YYMMDDw. informats

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

MDY function in Chapter 11, “SAS Functions”

DATEw., DDMMYYw., MMDDYYw., and YYMMDDw. formats in Chapter 14, “SAS Formats”

YEARCUTOFF= system option in Chapter 16, “SAS System Options”

Ew.d**Reads scientific notation**

Numeric

Width range: 7–32

Default width: 12

HELP INFORMAT

Description

The Ew.d informat reads numeric values that are stored in scientific notation. The w value specifies the width of the field containing the numeric value. The d value specifies the number of digits to the right of the decimal point in the numeric value. The d value range is 0 through 31.

Comparisons

The Ew.d informat is not used extensively because the SAS informat for standard numeric data, the w.d informat, can read numbers in scientific notation. Use the Ew.d informat to permit only scientific notation in your input data.

Example

Data Line	SAS Statement	Result
-----+-----1-----+-----2 1.257E3	input @1 x e7.;	1257

HEXw.**Converts hexadecimal positive binary values to fixed- or floating-point values**

Numeric

Width range: 1–16

Default width: 8

HELP INFORMAT

Description

The HEXw. informat reads hexadecimal digits and converts them to either integer binary (fixed-point) or real binary (floating-point) values. The w value specifies the field width of the input value, and also specifies whether the final value is fixed-point or floating-point.

When the w value is less than 16, the HEXw. informat converts input to positive integer binary values, treating all input values as positive (unsigned).

When the w value is 16, the HEXw. informat converts input to real binary (floating-point) values, including negative values.

Note: Different computer systems store floating-point values in different ways. However, the HEX16. informat reads hexadecimal representations of floating-point values with consistent results if the values are expressed in the same way your computer system stores them.

The HEXw. informat ignores leading or trailing blanks.

Example

Data Line *	SAS Statement	Result
-----+-----1-----+-----2 88F 4152000000000000	input @1 x hex3. @5 y hex16.;	2191 5.125

* The data line shows IBM mainframe hexadecimal data.

IBw.d

Reads integer binary (fixed-point) values

Numeric

Width range: 1–8

Default width: 4

HELP INFORMAT

Description

The IBw.d informat reads integer binary (fixed-point) values, including negative values represented in two's complement notation. If you use a *d* value, the IBw.d informat divides the number by 10^d . The *d* value range is 0 through 10.

Note: Different computer systems store integer binary values in different ways. However, the IBw.d informat reads integer binary values with consistent results if the values are created on the same type of computer system you use to run the SAS System.

Comparisons

Ordinarily, it is not possible to key in binary data directly from a terminal, although many programs write data in binary. The following table compares integer binary notation in several programming languages:

Language	Integer Binary Notation	
	2 Bytes	4 Bytes
FORTRAN	INTEGER*2	INTEGER*4
C	short	long
PL/1	FIXED BIN(15)	FIXED BIN(31)
COBOL	COMP PIC 9(4)	COMP PIC 9(8)
IBM 370 assembler	H	F

Example

Data Line *	SAS Statement	Result
00000080	input @1 x ib4.;	128

* The data line is a hexadecimal representation of a 4-byte integer binary number. Each byte occupies one column of the input field.

JULIANw.

Reads Julian dates (yyddd or yyyyddd)

Date and time

Width range: 5–32

Default width: 5

HELP INFORMAT

Description

The JULIANw. informat reads Julian dates in the form *yyddd* or *yyyyddd*, where *yy* or *yyyy* is a two-digit or four-digit integer representing a specific year, and *ddd* is an integer from 1 through 365 (366 for a leap year), representing the day of the year. Julian dates consist of strings of contiguous numbers, which means that zeros must pad any space between the year and day value. The example below illustrates this point.

Note: Julian dates containing year values before 1582 are not valid for conversion to Gregorian dates.

Use the YEARCUTOFF= system option to override the system default and specify a date range of your choice.

Example

Data Line *	SAS Statement	Result
-----+-----1-----+-----2		
90091	input day julian8.;	11048

* The data line corresponds to the ninety-first day of 1990, or 01APR90.

See Also

JULIANw. informat
“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”
DATEJUL and JULDATE functions in Chapter 11, “SAS Functions”
JULIANw. format in Chapter 14, “SAS Formats”
YEARCUTOFF= system option in Chapter 16, “SAS System Options”

MMDDYYw.

Reads date values (mmddyy)
Date and time
Width range: 6–32
Default width: 6
HELP INFORMAT

Description

The MMDDYYw. informat reads date values in *mmddyy* form, where *mm*, *dd*, and *yy* are integers representing the month, day, and year. The month, day, and year fields can be separated by blanks or special characters. However, if delimiters are used, they should be placed between all fields in the value. Blanks can also be placed before and after the date.

Note: The SAS System defaults to a date in the 1900s if *yy* is two digits. Use the YEARCUTOFF= system option to override the system default and specify your choice of date range.

Example

Data Lines	SAS Statement	Results
-----+-----1-----+-----2		
010190	input day mmddyy8.;	10958
1/1/90		10958
01 1 90		10958

See Also

DATEw., DDMMYYw., and YYMMDDw. informats

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

DAY, MDY, MONTH, and YEAR functions in Chapter 11, “SAS Functions”

DATEw., DDMMYYw., MMDDYYw., and YYMMDDw. formats in Chapter 14, “SAS Formats”

YEARCUTOFF= system option in Chapter 16, “SAS System Options”

MONYYw.

Reads month and year date values (mmmyy)

Date and time

Width range: 5–32

Default width: 5

HELP INFORMAT

Description

The MONYYw. informat reads date values in the form *mmmyy*, where *mmm* is the first three letters of the month name, and *yy* or *yyyy* is an integer representing the year. The *mmm* and *yy* or *yyyy* values cannot be separated by blanks. A value read with the MONYYw. informat results in a SAS date value corresponding to the first day of the specified month.

Note: The SAS System defaults to a date in the 1900s if *yy* is two digits. Use the YEARCUTOFF= system option to override the system default and specify a date range of your choice.

Example

Data Line	SAS Statement	Result
-----+-----1-----+-----2		
jun89	input month monyy5.;	10744

See Also

DDMMYYw., MMDDYYw., and YYMMDDw. informats

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

MONTH and YEAR functions in Chapter 11, “SAS Functions”

DDMMYYw., MMDDYYw., MONYYw., and YYMMDDw. formats in Chapter 14, “SAS Formats”

YEARCUTOFF= system option in Chapter 16, “SAS System Options”

MSECw.**Reads TIME MIC values**

Date and time

Width range: 8

Default width: 8

HELP INFORMAT

Description

The MSECw. informat reads time values produced by IBM mainframe operating systems, converting the time values to SAS time values. The *w* value must be 8, because the OS TIME macro or the STCK System/370™ instruction on IBM mainframes each return an 8-byte value.

Comparisons

The MSECw. and TODSTAMPw. informats both read IBM time-of-day clock values, but the MSECw. informat assigns a time value to a variable, and the TODSTAMPw. informat assigns a datetime value. Use the MSECw. informat to find the difference between two MVS TIME values, with precision to the nearest microsecond.

Example

Data Line *	SAS Statement	Result
0000EA044E65A000	input btime msec8.;	62818.412122

* The data line is a hexadecimal representation of a binary 8-byte time-of-day clock value. Each byte occupies one column of the input field. The result is a SAS time value corresponding to 5:26:58.41 PM.

See AlsoTODSTAMPw. informat

NENGOW.**Reads Japanese date values (r.yyymmdd)**

Date and time

Width range: 7–32

Default width: 10

HELP INFORMAT

Description

The NENGOW. informat reads Japanese date values in the form *r.yyymmdd*, where *r* is a letter representing an emperor's reign: M(Meiji), T(Taisho), S(Showa), or H(Heisei). The period is optional, and *yy*, *mm*, and *dd* are integers representing the year, month, and day. The year, month, and day values can be separated by blanks or any nonnumeric character. However, if delimiters are used, place them between all fields in the value.

See Also

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”
NENGOW. format in Chapter 14, “SAS Formats”

Comparisons

The *w.d* informat is identical to the *BZw.d* informat, except that the *w.d* informat ignores trailing blanks in the numeric values. To read trailing blanks as 0s, use the *BZw.d* informat.

The *w.d* informat can read values in scientific E-notation exactly as the *Ew.d* informat does.

Example

Data Line	SAS Statement	Results
-----+---1-----+---2		
23 2300	input @1 x 6. @10 y 6.2;	23 23
23 2300		23 23
23 -2300		23 -23
23.0 23.		23 23
2.3E1 2.3		23 2.3
-23 0		-23 0

YYMMDDw.

Reads date values (*yymmdd*)

Date and time

Width range: 6–32

Default width: 6

HELP INFORMAT

Description

The *YYMMDDw.* informat reads date values in *yymmdd* form, where *yy*, *mm*, and *dd* are integers representing the year, month, and day. The month, day, and year fields can be separated by blanks or special characters. However, if delimiters are used, they should be placed between all fields in the value. Blanks can also be placed before and after the date.

Note: The SAS System defaults to a date in the 1900s if *yy* is two digits. Use the *YEARCUTOFF=* system option to override the system default and specify your choice of date range.

Example

Data Lines	SAS Statements	Results
-----+---1-----+---2		
900101	input beg yymmdd8.;	10958
90 1 1		10958
90-01-01		10958
90/1/1		10958
19900101		10958
-----+---1-----+---2		
19520101	input date yymmdd10.;	-2922
1884/10/16		-27469

DATEw.**Writes date values
(ddmmmyy)**

Date and time

Width range: 5–9

Default width: 7

Alignment: right

HELP FORMAT

Description

The DATEw. format writes SAS date values in the form *ddmmmyy*, where *dd* is an integer representing the day of the month, *mmm* is the first three letters of the month name, and *yy* or *yyyy* is the year.

Example

Value	SAS Statements	Results
		----+----1----+----2
10847	put day date5.;	12SEP
	put day date6.;	12SEP
	put day date7.;	12SEP89
	put day date8.;	12SEP89
	put day date9.;	12SEP1989

See Also

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

DATE function in Chapter 11, “SAS Functions”

DATEw. informat in Chapter 13, “SAS Informats”

DATETIMEw.d**Writes datetime values
(ddmmmyy:hh:mm:ss.ss)**

Date and time

Width range: 7–40

Default width: 16

Alignment: right

HELP FORMAT

Description

The DATETIMEw.d format writes SAS datetime values in the format *ddmmmyy:hh:mm:ss.ss*, representing a specific day, month, year, hour, minute, second, and decimal fraction of a second. The SAS System requires a minimum *w* value of 16 to write a SAS datetime value with the date, hour, and seconds. Add an additional two places to the width to return values with optional decimal fractions of seconds.

Note: The *d* value can range from 1 through 39, but must be less than the specified *w* value. If *w*–*d* is less than 17, the SAS System truncates the decimal values.

Example

Value	SAS Statements	Results
		—+----1----+----2
937192783	put event datetime7.;	12SEP89
	put event datetime12.;	12SEP89:03
	put event datetime18.;	12SEP89:03:19:43
	put event datetime18.1;	12SEP89:03:19:43.0

See Also

DATEw. and TIMEw.d formats
“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”
DATETIME function in Chapter 11, “SAS Functions”
DATEw., DATETIMEw., and TIMEw. informats in Chapter 13, “SAS Informats”

DAYw.

Writes day of month
Date and time
Width range: 2–32
Default width: 2
Alignment: right
HELP FORMAT

Description

The DAYw. format writes the day of the month from a SAS date value.

Example

Value	SAS Statement	Result
		-----+-----1-----+-----2
10919	put date day2.;	23

See Also

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

DDMMYYw.

Writes date values (ddmmyy)
Date and time
Width range: 2–10
Default width: 8
Alignment: right
HELP FORMAT

Description

The DDMMYYw. format writes SAS date values in *ddmmyy* form, where *dd*, *mm*, and *yy* are integers representing a specific day, month, and year. When the *w* value is from 2 to 5, the SAS System prints as much of the month and day as possible. When the *w* value is 7, the date appears as a two-digit year without slashes, and the value is right aligned in the output field.

Example

Value	SAS Statements	Results
		-----+-----1-----+-----2
11316	put date ddmmyy5.;	25/12
	put date ddmmyy6.;	251290
	put date ddmmyy7.;	251290
	put date ddmmyy8.;	25/12/90

See Also

DATEw., MMDDYYw., and YYMMDDw. formats
“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”
MDY function in Chapter 11, “SAS Functions”
DATEw., DDMMYYw., MMDDYYw., and YYMMDDw. informats in Chapter 13, “SAS Informats”

DOLLARw.d

Writes numeric values with dollar signs, commas, and decimal points

Numeric
Width range: 2–32
Default width: 6
Alignment: right
HELP FORMAT

Description

The DOLLARw.d format writes numeric values with a leading dollar sign and a comma separating every three digits of each value. The w value specifies the total width of the output field. The d value, which must be either 0 or 2, specifies whether to include decimal digits in the value. If the d value is 2, the DOLLARw.d format writes a decimal point and two decimal digits. If the d value is 0, the DOLLARw.d format does not write a decimal point or decimal digits.

The hexadecimal representation of the code for the dollar sign character (\$) is 5B on EBCDIC systems and 24 on ASCII systems. The monetary character these codes represent may be different in other countries, but the DOLLARw.d format always produces one of these codes. If you need another monetary character, you can define your own format with the FORMAT procedure. See Chapter 18, “The FORMAT Procedure” in the *SAS Procedures Guide, Version 6, Third Edition* for more details.

Comparisons

The DOLLARw.d format operates like the DOLLARXw.d format, but the DOLLARXw.d format reverses the roles of the decimal point and the comma. This convention is common in European countries.

The DOLLARw.d format operates exactly like the COMMAw.d format except that COMMAw.d format does not write a leading dollar sign.

Example

Value	SAS Statement	Result
1254.71	put @3 netpay dollar10.2;	-----+-----1-----+-----2 \$1,254.71

DOLLARXw.d

Writes numeric values with dollar signs, periods, and commas

Numeric

Width range: 2–32

Default width: 6

Alignment: right

HELP FORMAT

Description

The DOLLARXw.d format writes numeric values with a leading dollar sign and a period separating every three digits of each value. The w value specifies the total width of the output field. The d value, which must be either 0 or 2, specifies whether to include decimal digits in the value. If the d value is 2, the DOLLARXw.d format writes a comma and two decimal digits. If the d value is 0, the DOLLARXw.d format writes no comma or decimal digits.

The hexadecimal representation of the code for the dollar sign character (\$) is 5B on EBCDIC systems and 24 on ASCII systems. The monetary character these codes represent may be different in other countries, but the DOLLARXw.d format always produces one of these codes. If you need another monetary character, you can define your own format with the FORMAT procedure. See Chapter 18 in the *SAS Procedures Guide* for more details.

Comparisons

The DOLLARXw.d format operates like the DOLLARw.d format, but reverses the roles of the decimal point and the comma. This convention is common in European countries.

The DOLLARXw.d format operates like the COMMAXw.d format except that COMMAXw.d does not print a leading dollar sign.

Example

Value	SAS Statement	Result
1254.71	put @3 netpay dollarx10.2;	-----1-----2 \$1.254,71

DOWNAMEw.

Writes name of day of the week

Date and time

Width range: 1–32

Default width: 9

Alignment: right

HELP FORMAT

Description

The DOWNAMEw. format writes the name of the day of the week from a SAS date value, with the first letter capitalized and the remainder of the name in lowercase letters. The name of the day is truncated to fit the format width if necessary. Thus, the format DOWNAME2. can be used to print the first two letters of the day name. If no w value is specified, the entire name of the day is printed.

Example

Value	SAS Statement	Result
10621	put date downame.;	-----1-----2 Sunday

Example

Data Value	SAS Statement	Result*
128	put x ib4.;	00000080

* The result is a hexadecimal representation of a 4-byte integer binary number. Each byte occupies one column of the output field.

JULDAYw.

Writes Julian day of the year

Date and time

Width range: 3—32

Default width: 3

Alignment: right

HELP FORMAT

Description

The JULDAYw. format writes the Julian day of the year from a SAS date value.

Example

Value	SAS Statement	Result
10621 10624	put date1 julday3. +1 date2 julday3.;	-----+-----1-----+-----2 29 32

See Also

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

JULIANw.

Writes Julian dates (yyddd or yyyyddd)

Date and time

Width range: 5—7

Default width: 5

Alignment: left

HELP FORMAT

Description

The JULIANw. format returns a Julian date from a SAS date value. The SAS System writes the Julian date with a two-digit year when the w value is 5. If the w value is 7, the Julian year value is four digits. To avoid confusion, use a four-digit year when the century is not clearly defined.

Examples

Value	SAS Statements	Results
9952	put date julian5.;	-----+-----1-----+-----2 87091
	put date julian6.;	087091
	put date julian7.;	1987091

See Also

DATEJUL and JULDATE functions

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

JULIANw. informat in Chapter 13, “SAS Informats”

MMDDYYw.

Writes date values (mmddyy)

Date and time

Width range: 2–10

Default width: 8

Alignment: right

HELP FORMAT

Description

The MMDDYYw. format writes a SAS date value in *mmddyy* form, where *mm*, *dd*, and *yy* are integers representing the month, day, and year.

Examples

Value	SAS Statements	Results
		----+----1----+----2
10847	put day mmddyy2.;	09
	put day mmddyy3.;	09
	put day mmddyy4.;	0912
	put day mmddyy5.;	09/12
	put day mmddyy6.;	091289
	put day mmddyy7.;	091289
	put day mmddyy8.;	09/12/89

See Also

DATEw., DDMMYYw., and YYMMDDw. formats

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

DAY, MDY, MONTH, and YEAR functions in Chapter 11, “SAS Functions”

DATEw., DDMMYYw., MMDDYYw., and YYMMDDw. informats in Chapter 13, “SAS Informats”

MMSSw.d**Writes minutes and seconds**

Date and time

Width range: 2–20

Default width: 5

Alignment: right

HELP FORMAT

Description

The **MMSSw.d** format converts a SAS time value to the number of minutes and seconds since midnight. The SAS System requires a minimum *w* value of 5 to write a value representing minutes and seconds. If the optional *d* value is specified, the SAS time value includes fractional seconds. The *d* value can range from 1 through 19, but must be less than the *w* value.

Example

Value	SAS Statement	Result
		-----+-----1-----+-----2
4530	put time mmss.;	75:30

See Also

HHMMw.d and TIMEw.d formats

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

HMS, MINUTE, and SECOND functions in Chapter 11, “SAS Functions”

TIMEw. informat in Chapter 13, “SAS Informats”

MMYYxw.**Formats that write month and year, separated by a character**

Date and time

Width range: 5–32

Default width: 7

Alignment: right

HELP FORMAT

Description

The **MMYYxw.** formats write the month (01 through 12) and year from a SAS date value, separated by colons, dashes, or other specific characters. The value of *x* can be C, D, N, P, or S, representing the type of separator; *w* represents the format width value. If the *w* value is too small to print a four-digit year, only the last two digits of the year are printed.

Example

Refer to the following table for a list of **MMYYxw.** formats and examples returned from an original date value of 29may1989:

Name	Separator	Example	Results
MMYYw.	letter M	mmyy5.	05M89
MMYYCw.	colon	mmyyc7.	05:1989
MMYYDw.	dash	mmyyd7.	05-1989
MMYYNw.	no separator	mmyyn6.	051989
MMYYPw.	period	mmyyp6.	05.89
MMYYSw.	slash	mmyys7.	05/1989

See Also

YYMMxw. formats

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

MONNAMEw.

Writes name of month

Date and time

Width range: 1–32

Default width: 9

Alignment: right

HELP FORMAT

Description

The MONNAMEw. format writes the name of the month from a SAS date value, with the first letter capitalized and the remainder of the name in lowercase letters. The name of the month is truncated to fit the format width if necessary. Thus, the format MONNAME3. can be used to print the first three letters of the month name.

Example

Value	SAS Statements	Results
		----+----1----+----2
10919	put date monname9.;	November
	put date monname1.;	N

See Also

MONTHw. format

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

MONTHw.

Writes month of year

Date and time

Width range: 2–32

Default width: 2

Alignment: right

HELP FORMAT

Description

The MONTHw. format writes the month (01 through 12) of the year from a SAS date value.

Example

Value	SAS Statement	Result
		----+----1----+----2
10919	put date month.;	11

See Also

MONNAMEw. format

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

MONYYw.

Writes month and year
(mmmyy or mmmyyyy)

Date and time

Width range: 5—7

Default width: 5

Alignment: right

HELP FORMAT

Description

The MONYYw. format writes SAS date values in the form *mmmyy*, where *mmm* is the first three letters of the month name, and *yy* or *yyyy* is a two- or four-digit integer representing the year.

Example

Value	SAS Statement	Result
		-----+-----1-----+-----2
10750	put acquired monyy7.;	JUN1989

See Also

DDMMYYw., MMDDYYw., and YYMMDDw. formats
“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”
MONTH and YEAR functions in Chapter 11, “SAS Functions”
MONYYw. informat in Chapter 13, “SAS Informats”

NEGPARENw.d

Displays negative values in
parentheses

Numeric

Width range: 1—32

Default width: 6

Alignment: right

HELP FORMAT

Description

The NEGPARENw.d format displays negative numbers enclosed in parentheses and nonnegative numbers with blanks instead of parentheses for proper column alignment. That is, the NEGPARENw.d format reserves the last column for a right parenthesis, even when the value is positive. The NEGPARENw.d format uses minus signs if a field is not wide enough for a number with parentheses.

The *d* values, which must be either 0 or 2, specify whether to include decimal digits in the values. If you specify a *d* value of 2, the NEGPARENw.d format writes a decimal point and two decimal digits. If you specify a *d* value of 0, the NEGPARENw.d format does not write a decimal point or decimal digits.

Comparisons

The NEGPARENw.d format operates just like the COMMAw.d format, separating every three digits of the value with a comma.

Example

Values	SAS Statement	Results
		-----+-----1-----+-----2
1000	put a1 sales negparen10.;	1,000
-2000		(2,000)

NENGOW.

Writes Japanese dates
(*r.yymmdd*)

Date and time

Width range: 2–10

Default width: 10

Alignment: left

HELP FORMAT

Description

The NENGOW. format writes Japanese date values in the form *r.yymmdd*, where *r* is a letter representing an emperor's reign: M (Meiji), T (Taisho), S (Showa), or H (Heisei). The period is optional and *yy*, *mm*, and *dd* are integers representing the year, month, and day.

Examples

Value	SAS Statements	Results
		----+----1----+----2
9784 *	put date nengo2.;	61
	put date nengo3.;	S61
	put date nengo4.;	S.61
	put date nengo5.;	S6110
	put date nengo6.;	S61/10
	put date nengo7.;	S611015
	put date nengo8.;	S.611015
	put date nengo9.;	S61/10/15
	put date nengo10.;	S.61/10/15

* This corresponds to 15OCT86.

See Also

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”
NENGOW. informat in Chapter 13, “SAS Informats”

Example

Value	SAS Statement	Result *
12	put x pib1.;	0C

* The result is a hexadecimal representation of a 1-byte binary number written in positive integer binary format, occupying one column of the output field.

PKw.d

Writes unsigned packed decimal data

Numeric
Width range: 1–16
Default width: 1
Alignment: left
HELP FORMAT

Description

The PKw.d format converts numeric values to unsigned packed decimal values. Each byte of unsigned packed decimal data contains two digits. The w value specifies the width of the output field. When you specify a d value, the PKw.d format multiplies the number by 10^d.

Comparisons

The PKw.d informat operates like the PDw.d informat except that the PKw.d format does not write the sign in the low-order byte.

Example

Value	SAS Statement	Result *
128	put x pk4.;	00000128

* The result is a hexadecimal representation of a 4-byte number written in packed decimal format. Each byte occupies one column of the output field.

QTRw.

Writes quarter of year
Date and time
Width range: 1–32
Default width: 1
Alignment: right
HELP FORMAT

Description

The QTRw. format writes the quarter of the year from a SAS date value.

Example

Value	SAS Statement	Result
10741	put date qtr.;	-----1-----2

See Also

QTRRw. format
“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

QTRRw.

Writes quarter of year in Roman numerals

- Date and time
- Width range: 3–32
- Default width: 3
- Alignment: right
- HELP FORMAT

Description

The QTRRw. format writes the quarter of the year from a SAS date value using Roman numerals.

Example

Value	SAS Statement	Result
		----+----1----+----2
10897	put date qtrr3.;	IV

See Also

- QTRw. format
- “SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

RBw.d

Writes real binary (floating-point) data

- Numeric
- Width range: 2–8
- Default width: 4
- Alignment: left
- HELP FORMAT

Description

The RBw.d format writes numeric data in real binary (floating-point) notation. The w value specifies the width of the output field. When you specify a d value, the RBw.d format multiplies the number by 10^d, then applies the real binary format to that value. The d value is just a scaling factor and does not indicate the number of decimal places.

The RBw.d format writes numeric data in the same way the SAS System stores them. Since it requires no data conversion, the RBw.d format is the most efficient method for writing data with the SAS System.

Note: Different computer systems store real binary values in different ways. However, the RBw.d format writes real binary values with consistent results on the same kind of computer system you use to run the SAS System.

► **Caution:** Using RB4 may result in truncation.

Using RB4. to write real binary data on equipment conforming to the IEEE standard for floating-point numbers results in a truncated 8-byte number rather than a true 4-byte floating-point number.

Comparisons

The following table compares the names of real binary notation in several programming languages:

Language	Real Binary Notation	
	4 Bytes	8 bytes
SAS	RB4.	RB8.
FORTRAN	REAL*4	REAL*8
C	float	double
COBOL	COMP-1	COMP-2
IBM 370 assembler	E	D

See Also

TIMEw.d format

“SAS Date and Time Values” in Chapter 4, “Rules of the SAS Language”

TIMEPART function in Chapter 11, “SAS Functions”

TIMEw. informat in Chapter 13, “SAS Informats”

w.d

Writes standard numeric data

Numeric

Width range: 1–32

Alignment: right

HELP FORMAT

Description

The w.d format writes standard numeric values one digit per byte. The w value specifies the width of the output field. The d value optionally specifies the number of digits to the right of the decimal point in the numeric value. If you specify 0 for the d value or do not specify any d value, the w.d format writes the value without a decimal point.

The w.d format rounds to the nearest number that fits in the output field. If the number is too large to fit, the w.d format uses the BESTw. format. The w.d format writes negative numbers with leading minus signs. In addition, the w.d format right justifies before writing and pads with leading blanks.

When you are choosing a w value, remember to allow enough space to write the value, the decimal point, and a minus sign, if necessary.

Example

Value	SAS Statement	Result
		-----+-----1-----+-----2
23.45	put a7 x 6.3;	23.450

WEEKDATEw.

Writes day of week and date (day-of-week, month-name dd, yy)

Date and time

Width range: 3–37

Default width: 29

Alignment: right

HELP FORMAT

Description

The WEEKDATEw. format writes a SAS date value in the form *day-of-week, month-name dd*, and yy or yyyy. If the w value is too small to write the complete day of the week and month, the SAS System abbreviates as needed.

Comparisons

The WEEKDATEw. format is the same as the WEEKDATXw. format except the WEEKDATEw. format prints *dd* after the month's name.

Example

Value	SAS Statements	Results
		----+----1----+----2
10848	put begin weekdate3.;	Wed
	put begin weekdate9.;	Wednesday
	put begin weekdate15.;	Wed, Sep 13, 89
	put begin weekdate17.;	Wed, Sep 13, 1989

See Also

DATEw., DDMMYyw., MMDDYYw., TODw., WEEKDATXw., and YYMMDDw. formats

"SAS Date and Time Values" in Chapter 4, "Rules of the SAS Language"
JULDATE, MDY, and WEEKDAY functions in Chapter 11,
"SAS Functions"

DATEw., DDMMYyw., MMDDYYw., and YYMMDDw. informats in Chapter 13, "SAS Informats"

WEEKDATXw.

Writes day of week and date (day-of-week, dd month-name yy)

Date and time

Width range: 3-37

Default width: 29

Alignment: right

HELP FORMAT

Description

The WEEKDATXw. format writes a SAS date value in the form *day-of-week, dd month-name yy* or *yyyy*. If the w value is too small to write the complete day of the week and month, the SAS System abbreviates as needed (see the WEEKDATEw. format example).

Comparisons

The WEEKDATXw. format is the same as the WEEKDATEw. format except that the WEEKDATXw. format prints *dd* before the month's name.

Example

Value	SAS Statement	Result
		----+----1----+----2----+
10869	put begin weekdatx.;	Wednesday, 4 October 1989

See Also

WEEKDATEw., WORDDATEw., and WORDDATXw. formats

"SAS Date and Time Values" in Chapter 4, "Rules of the SAS Language"
TODAY and WEEKDAY functions in Chapter 11, "SAS Functions"

DATEw., DDMMYyw., MMDDYYw., and YYMMDDw. informats in Chapter 13, "SAS Informats"

WEEKDAYw.

Writes day of week

Date and time

Width range: 1-32

Default width: 1

Alignment: right

HELP FORMAT

Description

The WEEKDAYw. format writes the day of the week from a SAS date value (where 1=Sunday, 2=Monday, and so on).

Example

Value	SAS Statement	Result
		-----+-----1-----+-----2
10621	put date weekday.;	1

See Also

DOWNAMEw. format
"SAS Date and Time Values" in Chapter 4, "Rules of the SAS Language"

WORDDATEw.

Writes date with name of month, day, and year
(month-name dd, yyyy)

Date and time

Width range: 3-32

Default width: 18

Alignment: right

HELP FORMAT

Description

The WORDDATEw. format writes a SAS date value in the form month-name dd, yyyy. If the w value is too small to write the complete month, the SAS System abbreviates as needed.

Comparisons

The WORDDATEw. format is the same as the WORDDATXw. format except that the WORDDATEw. format prints dd after the month's name.

Example

Value	SAS Statements	Results
		-----+-----1-----+-----2
11212	put term worddate3.;	Sep
	put term worddate9.;	September
	put term worddate12.;	Sep 12, 1990
	put term worddate18.;	September 12, 1990
	put term worddate20.;	September 12, 1990

See Also

WORDDATXw. format
"SAS Date and Time Values" in Chapter 4, "Rules of the SAS Language"

DATE

Prints the date and time

Valid as part of: configuration file, OPTIONS statement, OPTIONS window, SAS invocation

HELP DATE

Syntax

DATE | NODATE

Description

The DATE system option controls whether the date and time that the SAS job began are printed at the top of each page of the SAS log and any print file created by the SAS System. (In display manager and interactive line mode sessions, the date and time appear only on procedure output.)

You can use the following forms of the DATE system option:

DATE	specifies to print the date and time.
NODATE	specifies not to print the date and time.

DBCS

Recognizes double-byte character sets (DBCS)

Valid as part of: configuration file, SAS invocation

HELP DBCS

Syntax

DBCS | NODBCS

Description

The DBCS system option specifies whether the SAS System recognizes double-byte character sets (DBCS). (Double-byte character sets use 2 bytes for each character in the set.)

The DBCS system option is used for various reasons including converting lowercase data that are input into the SAS System to uppercase and supporting languages such as Chinese, Japanese, Korean, and Taiwanese.

You can use the following forms of the DBCS system option:

DBCS	specifies that the SAS System process double-byte character sets.
NODBCS	specifies that the SAS System not process double-byte character sets.

See Also

DBCSLANG= system option

DBCSTYPE= system option

WORKTERM

Erases WORK files at the termination of a SAS session

Valid as part of: configuration file, OPTIONS statement, OPTIONS window, SAS invocation

Syntax

WORKTERM | NOWORKTERM

Description

The WORKTERM system option specifies whether SAS WORK files, such as data sets, are erased from the current SAS WORK data library at the termination of the SAS session. The aliases for this option are WRKTERM and NOWRKTERM.

You can use the following forms of the WORKTERM system option:

WORKTERM specifies to erase the WORK files.

NOWORKTERM specifies not to erase the WORK files.

Although NOWORKTERM prevents the WORK data sets from being deleted, it has no effect on initialization of the WORK library by the SAS System. The SAS System normally initializes the WORK library at the start of each session, which effectively destroys any pre-existing information.

Comparisons

Use the WORKINIT system option to control whether the WORK data library is cleared when the SAS System is invoked.

See Also

WORKINIT system option

YEARCUTOFF=

Specifies the first year of a 100-year span used by informats and functions

Valid as part of: configuration file, OPTIONS statement, OPTIONS window, SAS invocation

HELP YEARCUTOFF

Syntax

YEARCUTOFF=*nnnn* | *nnnnn*

Description

The YEARCUTOFF= system option specifies the first year of a 100-year span used as the default by various DATE and DATETIME informats and functions.

You can use the following argument with the YEARCUTOFF= system option:

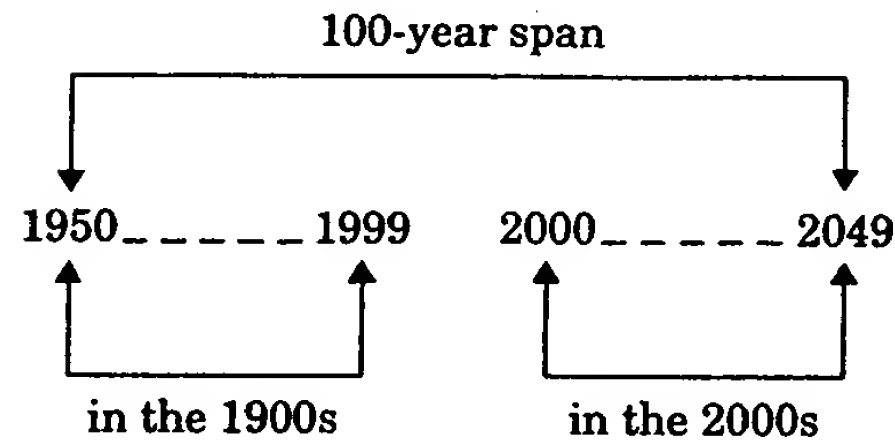
nnnn specifies the first year of the 100-year span. Valid values are
nnnnn from 1582 through 1990.

If the default value of *nnnn* (1900) is in effect, the 100-year span begins with 1900 and ends with 1999. Therefore, any informat or function that uses a two-digit year value assumes a prefix of 19. For example, the value 92 refers to the year 1992.

Note that the value specified in the YEARCUTOFF= system option can result in years that occur in two centuries. For example, if you specify YEARCUTOFF=1950, any two-digit value between 50 and 99 inclusive refers to the first half of the 100-year span, which is in the 1900s. Any two-digit value between 00 and 49 inclusive refers to the second half of

the 100-year span, which is in the 2000s. Figure 16.1 illustrates the relationship between the 100-year span and the two centuries if YEARCUTOFF=1950.

Figure 16.1 A 100-year Span with Values in Two Centuries



The YEARCUTOFF= system option applies to one- and two-digit years specified in the MMDDYY, YYMMDD, MONYY, DDMMYY, DATE, DATETIME, and YYQ informats and to one- and two-digit years specified in the DATEJUL, MDY, MONYY, and YYQ functions.

■ Host Information

The syntax shown above applies to the OPTIONS statement. However, when you specify the YEARCUTOFF= system option on the command line or in a configuration file, the syntax is host specific and may include additional or alternate punctuation. For details, refer to the SAS documentation for your host system.

..... ■

COMPUTERWORLD

Oracle Office to throw down the gauntlet to Notes

Groupware blueprint adds image, voice to Oracle DBMS

By Kim S. Nash
REDWOOD SHORES, CALIF.

Oracle Corp. plans to disclose blueprints for a groupware product a la Lotus Notes during its annual International Oracle User Group conference later this month. The database maker is also expected to unveil several so-called "media server" add-on modules that will let its Oracle database handle voice, images and full-motion video, sources said.

Not content to let Lotus Development Corp. control a burgeoning groupware market, Oracle plans to eventually remake Oracle Office into a Notes rival, according to analysts and users familiar with Oracle's product plans.

Oracle Office is a set of electronic-mail, scheduling and directory service applications linked to the Oracle relational database. A groupware version of Oracle Office is expected to be delivered in mid- to late 1994, sources said.

Larry Ellison: Plans multimedia demo

Lotus and Oracle have discussed linking Notes with the Oracle database, analysts said, but a revamped Oracle Office would put the database maker in competition with Lotus.

Oracle, page 24

IBM Office struggles

Key accounts defect despite new OfficeVision initiatives, products

By Johanna Ambrosio

IBM is fighting to keep OfficeVision — its much-delayed and maligned office automation framework — alive even as some key customers are leaving the fold.

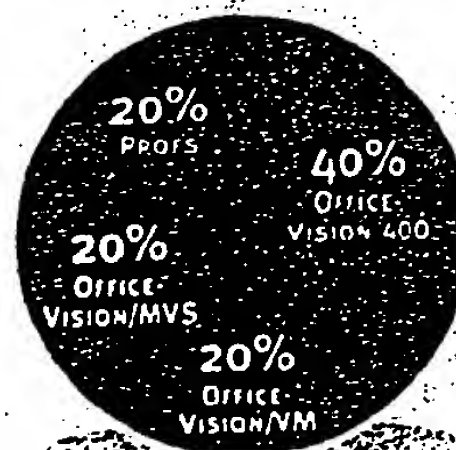
For example, IBM's recent introduction of some OfficeVision components that run on local-area networks (see story page 16) has not been enough to stop some large accounts from kissing the whole thing good-bye. Southern California Edison Co., a huge utility in Rosemead, Calif., will by early next year move its 10,000 OfficeVision end users to an as-yet undetermined alternative system.

"OfficeVision just does not meet our long-term direction for client/

Mixing it up

According to IBM reports, the number of OfficeVision end users is growing by 14% annually

BREAKDOWN OF 9 MILLION OFFICEVISION END USERS WORLDWIDE



server," said David Tommela, manager of information technologies. He said the utility looked into the IBM LAN offerings but decided to go with another supplier. Whichever environment it chooses will eventually support 17,000 people.

Despite these defections, Richard Sullivan, IBM's director of office and publishing marketing, insisted that "OfficeVision is alive and well and still growing." He claimed the user base is growing by approximately 14% annually, but he did not break out what percentage of that figure was new accounts vs. nodes added to existing OfficeVision installations. Overall, IBM claims some 5 million end

OfficeVision, page 16



IBM says most of the new account growth in OfficeVision comes from the AS/400. About 40% of all AS/400s going out the door have OfficeVision installed.

Looming specter of NT unifying Unix

By Jean S. Bozman

The alignment last week of more than 75 Unix vendors behind a common set of Unix application programming interfaces fired a broadside at an old foe, Microsoft Corp., whose Windows NT operating system is just moving into the early stages of delivery.

However, users and analysts cautioned that the alliance's impact may be lessened by a mid-1994 delivery schedule.

"Microsoft has done more to unify Unix than 20 years of history," — Alan Fedder, executive director of Washington, D.C., Area Unix Users Group

The weapon wielded by the Unix vendors at their New York announcement last week was economics (see story page 8).

A common set of APIs will reduce the cost of creating packaged Unix applications, making it easier and cheaper for developers such as Lotus Development Corp., Borland International, Inc. and Oracle Corp. to write to the Unix client/server environment. This is critical if Unix is to compete with Unix/NT war, page 9

IBM PC Co. wins users; backlog threatens gains

By Michael Fitzgerald
NEW YORK

As IBM PC Co. marks its first anniversary, analysts and many users said the independent company

has erased much of the stigma associated with the IBM name. But all sides agreed the PC Co. still has lots to do before it makes its way back to the head of the class.

Its most pressing piece of homework is to



IBM report card

ROBERT CORRIGAN, president of the IBM PC Co., gave his company an A for organization and a C for delivery. Analysts were generally more positive, giving the PC Co. a solid A-/B+ for becoming proactive.

work is to end its relentless backlog problems. And then there is the need to pack more users into a cheering section currently weighted more heavily in the analyst community.

Results from a Computerworld survey of 100 information systems managers last week indicated that while many users no longer see IBM's PC effort as uncompetitive, a good many have made it only to neutral ground and are still reserving judgment.

There is no question that an unshackled IBM PC Co. is serving as an example to the rest of its struggling parent's units.

"They've taken a business that was spiraling downward and revitalized it," said Richard Zwetch-PC Co., page 14



SEVENTH ANNUAL SPECIAL REPORT ON PAY TRENDS

WOMEN are making inroads into higher-paying, more responsible jobs in the traditionally male-dominated field of information systems.

But don't cheer yet: Female IS managers still earn an average of 15% less than their male counterparts — a distinct improvement over the 21% shortfall posted last year but still a telling testament to the gender gap.

IS compensation overall rose an average of 3% to 5% this year. See page 91.

BETTER BUT NOT GREAT

How women's pay compares with men's in the same job

Computerworld	6.2%
IBM	15.4%
Oracle	10.9%
Lotus	+6.1%

At the Crossroads: What's Next for Standard C?

18

Volume 12, Number 10
October 1994

C/C++ Users Journaltm

Advanced Solutions for C/C++ Programmers

Debugging

• Exotic Assert Macros in C++

With Classes — A Serious Proposal

Exploring C++'s Strange New Pointers

More Date Math and a Type date_t

Columns By:

• P.J. Plauger

• Dan Saks

• Ken Pugh

• Victor R. Volkman



32808 74098



C/C++ Users Journaltm

Advanced Solutions for C/C++ Programmers

DEBUGGING

Powerful Assertions for C++

By *Harald M. Mueller*

21

If you believe in assertions to enforce pre- and post-conditions, here's a C++ header file that gives you lots of machinery for writing complex assertions

FEATURES

All is Flux

By *Bob Jervis*

39

C++ was originally called "C with Classes." The author of Borland's Turbo C revisits those roots and takes a new departure.

Powerful Pointers to Member Functions

By *Christopher Skelly*

51

So you thought you finally understood pointers in C. Now meet pointers to members in C++.

An Extended Date Library for C

By *Stan Milam*

67

If you need to manipulate dates spanning several centuries, this suite of functions nicely complements those in the Standard C library.

Cover design by *Susan Schuette Buchanan*

C/C++ Users Journal (ISSN 1075-2838) is published monthly by R&D Publications, Inc., 1601 W. 23rd St., Suite 200, Lawrence, KS 66046-2700 (913) 841-1631. Second-class postage paid at Lawrence, KS and additional mailing offices. POSTMASTER: Send address changes to C/C++ USERS JOURNAL, 1601 W. 23rd St., Suite 200, Lawrence, KS 66046-2700. Subscriptions: Annual renewable subscriptions to C/C++ Users Journal are \$29.95 US, \$46 Canada and Mexico, \$65 overseas. Payments must be made in US dollars. Make checks payable to C/C++ Users Journal. GST (Canada): #129065819

COLUMNS

Standard C: Inserters

P. J. Plauger

10

Stepping Up to C++:

Designing Generic Container Classes, Part 4

Dan Saks

81

Questions & Answers: Handling Constructor Failures

Kenneth Pugh

93

Code Capsules: Dynamic Memory Management, Part 1

Chuck Allison

101

CUG New Releases: Bison ++/Flex++ Update, Saltsoft Tools, LIBFTP, and RasMol

Victor R. Volkman

119

DEPARTMENTS

Editor's Forum

8

Product Pages

63

InstantInfo/Advertiser Index

96

Calendar of Events

100

Call for Papers

122

New Products

123

We Have Mail

127

Programmer's Market

129

CUG Online Source Code

You can obtain all code published (and some unpublished) in CUG from:

USENET (Archived by UUNET Technologies 1-800-488-6384):

uunet!~published/cuj/19YY/monYY.tar.Z

Available via anonymous FTP from <ftp.uu.net> or via uucp from (900)GOT-SRCS.

Download via uucp and uncompress using `compress-d filename`, then `tar xvf filename` to expand the archive.

BBS:

The Courts of Chaos, 501-985-0059; EmmaSoft Shareware Board, 607-533-7072; Phoenix Chapter ACM Library, 602-970-0474; The Programmer's Corner, 301-596-7692; C_BBS (The Netherlands), +31-(0)-4930-20361 or +31-(0)-4930-20792.

CompuServe:

GO SDFORUM, Library 7 (R&D Publications)

GEnie:

In the IBM-PC Roundtable at page 615 (Keyword: IBMPC).

Monthly Code Disk:

C/C++ Users Journal™

Advanced Solutions for C/C++ Programmers

EDITORIAL

Publisher

Robert Ward

Senior Editor

P. J. Plauger

Managing Editor

Marc Briand

Contributing Editors

Chuck Allison

Steve Graham

Kenneth Pugh

Dan Saks

Victor R. Volkman

Sydney Weinstein

CUSTOMER SERVICE

Customer Service

Pam VanSchmus

Jodi Leonard

913-841-1631

FAX 913-841-2624

ADVERTISING AND MARKETING

Marketing Director

Jeff Dickey-Chasins

Acct. Manager, East

Ed Day

913-841-1622

Acct. Manager, Midwest

Christine Woodley

913-841-1633

Acct. Manager, West

Edwin Rolhock

913-841-1626

European

Advertising

Representative

Brian Osborn

breakout marketing

Friedrichsorter Str. 7

D-24159 Kiel

(Germany)

+49 431-396895

FAX +49 431-396827

Direct Marketing

Bill Uhler

PRODUCTION

Art Director

Susan Schuette Buchanan

Graphic Artist

Twyla Watson Bogard

Production Editor

Amy Pettit

Advertising Materials

Lori White

C/C++ Users Journal is the successor to the C Users' Group Newsletter and The C Journal. Subscribers are automatically enrolled as members of The C Users' Group. C/C++ Users Journal and The C Users' Group are services of R&D Publications, Inc., Lawrence, KS.

Entire contents Copyright ©1994 R&D Publications, Inc. No portion of this publication may be reproduced, stored, or transmitted in any form, including computer retrieval, without written permission from the publisher. All Rights Reserved. Quantity reprints of selected articles may be ordered. By-lined articles express the opinion of the author and are not necessarily the opinion of the publisher.

Printed in the United States of America.

Advertising: For rate cards or other information on placing advertising in C/C++ Users Journal, contact the advertising department at (913) 841-1631, or write C/C++ Users Journal, 1601 W. 23rd St., Ste. 200, Lawrence, KS 66046-2700.

Customer Service: For subscription orders and address changes, contact C/C++ Users Journal, 1601 W. 23rd St., Ste. 200, Lawrence, KS 66046-2700. Telephone (913) 841-1631; FAX (913) 841-2624; e-mail cujsub@rpub.com.

Trademarks: C/C++ Users Journal, R&D Publications, Inc., UNIX, AT&T Bell Laboratories, XENIX, MS-DOS, OS/2, Microsoft C and QuickC, Windows, Microsoft Corporation, IBM, IBM-PC, PS/2, International Business Machines Corp., Brief, Turbo C, Turbo C++, Borland International, Macintosh, Apple Computer, Inc., Zortech C++, Zortech VAX-VMS, Digital Equipment Corp., DR-DOS, Digital Research, DESQview, Quarterdeck, Atari, Atari, Inc., Amiga, Commodore, Inc.

An Extended Date Library for C

Stan Milam

Introduction

One of the most convenient attributes of the Standard C library time and date functions is their relatively fine granularity, which is in seconds. Having one-second resolution allows date and time to be stored as a single field, which is wonderful for logging purposes. The downside of this granularity is that it limits the range of the date and time functions. Most libraries use the *long* data type to store the combined date and time in the number of seconds elapsed since 1 January, 1970. As a result, standard functions cannot handle dates before 1 January, 1970. Also, since the *long* data type is 32 bits on most machines, the upper limit of the standard functions will be 18 January, 2038.

This upper limit is fine for most date calculations, and perhaps the *long* data type will someday be extended to 64 or more bits, but some date calculations need to exceed this limit now.

In this article I describe a suite of functions that is as versatile as the Standard C time/date functions, but have a greatly extended range. To accomplish both versatility and extended range I've mimicked the standard time/date functions, but raised the granularity of the stored time from elapsed seconds to elapsed days. For example, the standard *time()* function returns a value of type *time_t*, which contains the number of seconds elapsed since 1 January, 1970. I've written a corresponding function called *date()* which returns the number of days elapsed since 1 January, 0001 A.D. I've also written several functions which do not correspond to any function in the Standard C library, but extend the capabilities of my date functions. Because my date functions are similar in nature to the standard functions, these extended functions could be easily adapted to work with standard time/date functions.

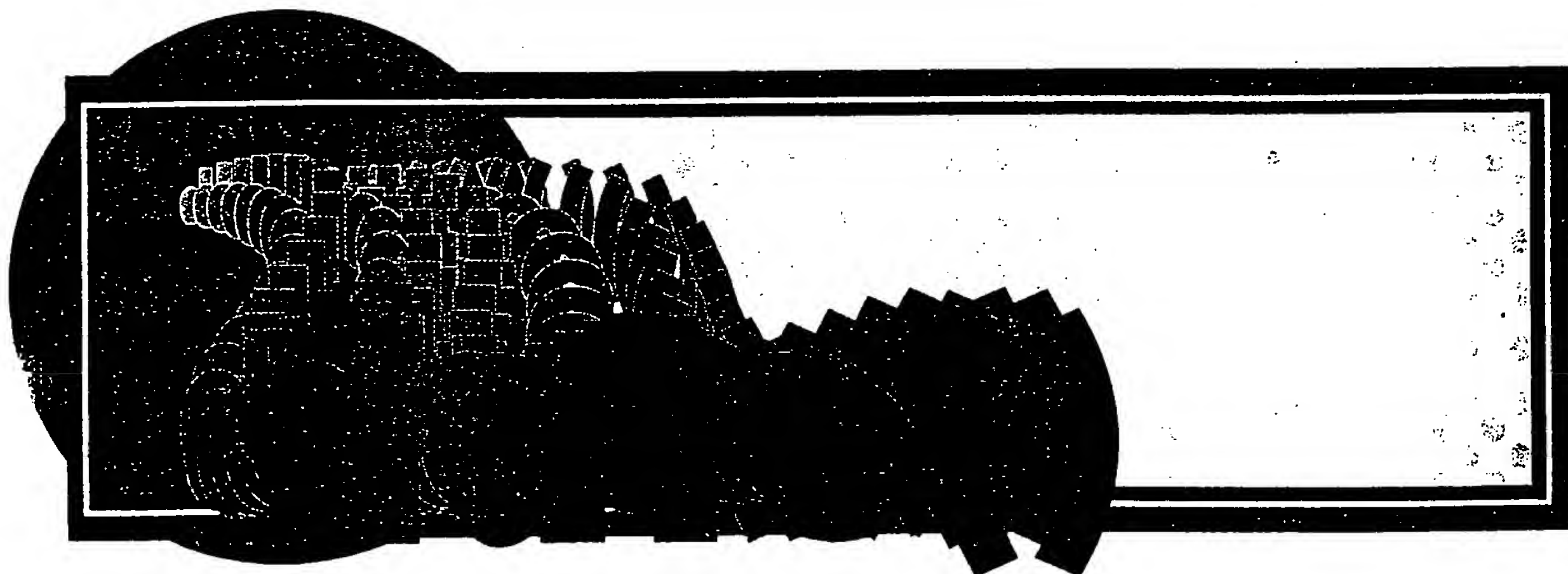
The Date Types

My date functions work with one of two date types which are analogous to the two time types in the Standard C library. These new date types are *date_t* (a *long*), and a structure, *struct dt*. The following code fragment shows how these new types are declared in the *dates.h* header file (this file is not listed here, but is included on this month's code disk):

```
typedef long date_t;    /* Sequential day value */
struct dt {
    int dt_leap_year;    /* Indicates a leap year */
    int dt_year;         /* The actual year */
    int dt_month;        /* The month, 0 - 11 */
    int dt_mday;         /* The day of month, 1-31 */
    int dt_yday;         /* The day of the year, 0-365 */
    int dt_wday;         /* The day of the week, 0-6, */
                        /* 0=Sun */
};
```

You can use *date_t* to define variables which represent a given date (uniquely) as a long integer. Because dates are usually comprised of three different values, (years, months, and days), representing a date as a long integer greatly simplifies date math. (To see how this integer is calculated, see the sidebar "Converting a date to a *date_t*".)

The second date type, *struct dt*, corresponds nicely to the structure used in the standard time/date functions, *struct tm*. *dt* includes only members which represent dates — hours, minutes, and seconds are not included. *dt* includes one additional member



Stan Milam is a C programmer working in the Dallas area. He also teaches C at Mountain View College in Dallas. He can be reached through his internet address: milam@metronet.com.

which does not correspond to any member of the time structure. This member, *dt_leap_year*, is set to 1 when *year* is a leap year and set to 0 otherwise. This structure is available for the programmer's use and is also used extensively by the date functions themselves.

The Date Functions

The date functions correspond closely to the standard time/date functions. However, they do not necessarily mimic the standard functions in every last detail. I will discuss the *date* function first since it is a building block for other functions. *date* (Listing 2)

Converting a Date to a *date_t*

Many of my date functions use parameters of type *date_t*, which is a *long* representing days elapsed since 1 January, 0001 A.D. This conversion occurs in three stages. First, all years previous to the current year must be converted into the days elapsed since 1 January, 0001 A.D. Function *years_to_days* (Listing 1) calculates this value by following the rules that determine leap years. Since there are always at least 365 days in a year, *years_to_days* multiplies the year value (current year - 1) by 365. It then adds the result of this operation to (year value) divided by 4 to account for leap years. Next, *years_to_days* subtracts (year value) divided by 100 to account for the century years which are not leap years. Finally, *years_to_days* adds (year value) divided by 400 to add back the centuries which were leap years. Thus, the formula is:

$$\text{date_value} = \text{year} * 365 + \text{year} / 4 - \text{year} / 100 + \text{year} / 400;$$

Users typically enter dates in terms of years, months, and days; these dates must be converted to type *date_t* before the date functions can use them. This value presumes the Gregorian Calendar in use today was implemented on 1 January, 0001 A.D., (which of course, it was not) and totally ignores the adjustments made by Pope Gregory in 1582 to the calendar. Ignoring this adjustment invalidates all calculations for dates prior to October of 1582, but works for later dates, since days elapsed merely serves as an index — its absolute value is not important.

In the second step, function *months_to_days* (Listing 1) uses a lookup table to convert all months prior to the current month into elapsed days. These days must be added to the value obtained in the first step. The final step is to simply add the current day of month to the value obtained from the first two steps.

Using this scheme, a huge range of dates is possible; however, I have imposed an artificial upper limit of 31 December, 9999 (which falls on a Friday).

DON'T TYPE IT

...JUST READ IT

Add a code disk subscription to your magazine subscription and get 12 disks (one each month) of *C/C++ Users Journal* listings.

SAVE

- o Hours of typing long code
- o yourself from entering errors
- o 50% off the price of individual disks

C/C++ Users Journal™

Advanced Solutions for C/C++ Programmers

CODE DISK SUBSCRIPTION

Only \$30 (prepaid) a year
\$50 (Non North American)

For more information or to order:

CALL 913-841-1631
FAX 913-841-2624

Listing 1 Supporting functions for date library

```

/*****
/*
/*      (c) Copyright 1993 by Stan Milam.
/*
/*
/*****
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <ctype.h>
#include "dates.h"

#define MAXDATE 3652059L      /* 31-Dec-9999 */

/*****
/* A table which contains an accumulation of days of all pre-
/* ceding months.
/*
/*****

static int month_accum_table[] = {
    0, 31, 59, 90, 120, 151,
    181, 212, 243, 273, 304, 334
};

/*****
/* Each element contains the normal number of days for each
/* month.
/*
/*****

static int month_table[] = {
    31, 28, 31, 30, 31, 30,
    31, 31, 30, 31, 30, 31
};

);

/*****
/* The full names of the weekdays.
/*
/*****

static char *full_weekday[] = {
    "Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday", "Friday", "Saturday"
};

/*****
/* The full names of the months.
/*
/*****

static char *full_month[] = {
    "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"
};

static int compare( char *s1, char *s2, unsigned len ) {

    unsigned ch1, ch2;

    while ( *s1 && *s2 && len ) {
        ch1 = toupper(*s1);
        ch2 = toupper(*s2);
        if (ch1 < ch2)
            return -1;
        else if ( ch1 > ch2 )
            return 1;
        else s1++, s2++, len--;
    }
};

```

SQL DATABASE ENGINE FOR C PROFESSIONALS

Give your programs a powerful database at the lowest cost
with the Just Logic database engine.

Just Logic is meant for professionals who write simple to very complex C/C++ systems and who need a true database without compromising performance.

The Just Logic database engine delivers:

- A clear and easy to read manual
- Comprehensive examples
- ANSI SQL
- Compatibility with popular compilers
- C and C++ interfaces
- RUNTIME LICENCE INCLUDED
- 30-day risk free money back guarantee

Limited time offer

DOS and Windows version	\$99
OS/2 version	\$99
DOS, Windows and OS/2 version	\$149

New products

SCO Unix	\$595
BSD Unix	\$295
Coherent	\$129

Call now: (800) 267-6887 (toll-free USA and Canada)
(800) 234-0141 (InstantInfo Doc #1185)
(514) 761-6887 (International)
(514) 642-6480 (Fax)

JUST LOGIC
TECHNOLOGIES

P.O. Box 63050,
40 Commerce St.,
Nun's Island, Que, Canada,
H3E 1V6

Request Reader Service #151/FAX #1185

Conversion Solutions for C and C++ Programmers

■ FOR C®

Converts standard FORTRAN and many VAX, PRIME and IBM-VS extensions into ANSI C! Features C-style preprocessing for easy mapping of FORTRAN library calls, C prototyping for enhanced function call translations, basic static analysis and error checking plus compiler-like performance. Translated code is extremely readable and maintainable!

■ FOR C++®

The only FORTRAN to C++ translator available! Translations support C++ string and complex data classes for good code readability. Generates C++ function prototypes, checks function calls for consistent usage, optimizes code for easy maintenance, and uses overloaded functions for readability. C++ is an excellent translation match for FORTRAN—and delivers more programming power!

■ Call today for more information
and special product discounts!

COBALT BLUE

COBALT BLUE, INC.
875 OLD ROSWELL ROAD, SUITE D-500
ROSWELL, GA 30076, USA
TEL (404) 518-1118, FAX (404) 840-1182

Request Reader Service #105/FAX #1157

Listing 1 continued

```

    }
    if ( len == 0 ) return 0;
    else if (*s1) return 1;
    else return -1;
}

static int is_it_a_leap_year( unsigned year ) {
    return ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0);
};
}

static date_t years_to_days( unsigned year ) {
    date_t rv;

    if (year > 0) year--;
    rv = year * 365L + year / 4L - year / 100L + year / 400L;
    return rv;
}

static date_t months_to_days( int month, int leap_year ) {
    date_t rv;

    rv = month_accum_table[month - 1];
    if ( month > 2 ) rv += leap_year;
    return rv;
}

static int days_to_months( int *days, int leap_year ) {

```

corresponds to the *time* function in the standard library. *date* returns the current date as type *date_t*. *date* actually calls the standard *time* function to get the current date/time value; it then passes the date/time value to a global utility function, *time_to_date*. *time_to_date* (Listing 1) produces a date value which function *date* then returns. This value can be used for date math, or as an argument to other date functions, including the extended date functions.

localdate (Listing 3) is one of the date functions that takes a *date_t* type as an argument. This function corresponds to the standard library's *localtime* function. *localdate* returns a pointer to an internal static structure of type *struct dt*, with members filled in to represent the date which was passed as an argument. This structure can also be used for date math, but it is more useful as an argument to other date functions. Since the extended date functions make heavy use of *localdate* you should make a copy of the returned structure value before calling any other functions.

Example:

```

struct dt dt;
dt = *localdate ( &date_value );

```

The *mkdate* (Listing 4) function is the inverse of the *localdate* function and corresponds to *mktime* in the standard library. *mkdate* converts a date structure back to a single date value of type *date_t*. *mkdate* uses the *dt_year* member, and attempts to use the *dt_month* and *dt_mday* members to calculate a date value. If the *dt_month* and *dt_mday* values are not available (or not valid), *mkdate* uses the *dt_yday* value to calculate the date. This behavior departs radically from *mktime*'s. *mkdate* differs from *mktime* in another way: unlike *mktime*, *mkdate* does not fix member values in the date structure if it finds them to be incorrect. (I implemented this adjustment early on, but removed it for the sake of efficiency.) If the values in the date structure are unsuitable for calculating a date value, *mkdate* returns a value of *(date_t)(-1)* to indicate an error.

Another function that accepts a date structure as an argument is the extremely useful *strfdate* (Listing 5). This function corresponds to the *strftime* function in the standard library. You use *strfdate* to build regular text strings of date values. *strfdate* uses format specifiers in a manner similar to *printf*. For example, when you call *strfdate*, *%A* is replaced with the full name of a weekday, *%B* is replaced with the full name of the month, and *%Y* is replaced by a four-digit year. *strfdate*'s format specifiers are exactly the same as *strftime*'s, except that *strfdate* does not recognize specifiers for time values.

What has your I/O library done to you lately?

FOR

Ordinary I/O libraries offer a number of standard features. Unfortunately, problems such as file-offset errors, cryptic file formats and data access errors are standard as well.

Introducing Gamelon, a new kind of I/O library. Its object-based structure eliminates the problems that are common with standard I/O libraries. Gamelon offers rapid file prototyping to speed application design, thread-safe file access to ease the transition to multi-threaded code, and a host of other features and tools that will save you time and money.

What *has* your I/O library been doing for you? To find out what an I/O library *can* do, contact Menai™ Corporation.

C and C++

gamelon™

Menai Corporation

1010 El Camino Real, Suite 370, Menlo Park, CA 94025 • info@menai.com
VOX 415.617.5730 • FAX 415.853.6453 • BBS 415.617.5726

Listing 1 *continued*

```

int rv, month, save_month;

save_month = month_table[1];
month_table[1] += leap_year;
for ( month = 0 ;; month++ ) {
    /* Go find the month and day */
    /* More days than in month? */
    if ( *days > month_table[month] ) {
        /* Yes, subtract from days */
        *days -= month_table[month];
    }
    else {
        /* Found month, days left over */
        rv = month;
        break;
    }
    /* Get out of loop */
}
month_table[1] = save_month;
return rv;
/* Return month */

/*****
/*      week_of_year()
/*
/* NOTE! This function was borrowed from P. J. Plauger's book
/* "The Standard C Library".
/*
*****/

static int week_of_year( int start, int wday, int yday ) {

    wday = ( wday + 7 - start ) % 7;
    return ( yday - wday + 12 ) / 7 - 1;
}

date_t time_to_date( time_t tv ) {

    date_t rv;
    struct tm *tm;
    int year, leap_year;

    /*****
    /* Get a time structure to use for conversion process.
    *****/

    tm = localtime(&tv);

    /*****
    /* Use values in the tm structure to convert the current
    /* date into a long integer value.
    *****/

    year = tm->tm_year + 1900;
    leap_year = is_it_a_leap_year(year);
    rv = years_to_days( year );
    rv += months_to_days(tm->tm_mon + 1, leap_year );
    rv += tm->tm_mday;

    return rv;
}

/* End of File */

```

PINNACLE RELATIONAL ENGINE

The Portable Compact Client-Server RDBMS
For C/C++ Programmers
DOS, Windows, UNIX, Mac, OS/2, NT, VMS

Painless Database Programming!

Vermont Database Corporation

1-800-822-4437

Vermont Database Corporation / 400 Upper Hollow Hill Road
Stowe, VT 05672 USA
802-253-4437 / 802-253-4146 (FAX)

□ Request 179 on Reader Service Card □

TLIBTM Version Control For DOS, OS/2 and Windows-NT

• The experts loved TLIB 4:

"...amazingly fast... TLIB is a great system." PC Tech Journal
"TLIB has features and power to spare... TLIB is easy to use and the fastest of the reviewed packages." Computer Language
"I will not program without it." Uptime Magazine

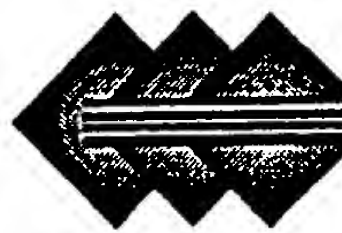
• TLIB 5.01 adds:

Automatic branching. Automatic version labeling across branches. User defined *promote* structures, for staged development. Exclusive whole-level *change migration* for customized software. N-way-tree version numbers. Branch and full locking. OS/2 & NT support. And now... automated conversion from PVCSTM or MKS RCS!

• Plus the features they loved in TLIB 4:

Check-in/out locking. Branching, for parallel development. Keywords. Full binary file support (does not depend upon NLS in the file like other products). Wildcard and list-of-file support; can create lists by scanning source code for includes. Can merge (reconcile) multiple simultaneous changes and undo intermediate revisions. Network and WORM optical disk support. Mainframe-compatible delta generator for Pansophic, ADR, IBM, Sperry formats. Integrated with industry-leading MAKE from OpusTM software.

MS-DOS \$139, OS/2 & NT (with MS-DOS) \$195 + shipping.
5-user net: DOS \$419, OS/2+NT+DOS \$595. Call for other sizes.



Burton Systems Software

PO Box 4157, Cary, NC 27519 (919) 233-8128
FAX: 233-0716

□ Request Reader Service #430/FAX #1186 □

The Extended Date Functions

As powerful as the regular date functions are, it becomes very tedious work doing date math, building date strings, and parsing date strings. For this reason I've written three extra functions. The first two functions are complementary and deal with date strings.

Listing 2 Function date

```

/*****
/*      (c) Copyright 1993 by Stan Milam      */
/*      *****/
/*****

date_t date( date_t *arg ) {

    date_t rv;

    /*****
    /* Use time_to_date() which uses the ANSI time() function */
    /* to do the actual date conversion.                      */
    /*****

    rv = time_to_date( time( NULL ) );

    /*****
    /* If an argument is supplied put the value in it.        */
    /*****

    if ( arg != NULL ) *arg = rv;
    return rv;
}

/* End of File */

```

Date strings usually occur in a limited number of formats; Building a "wrapper" function for *strfdate* to handle common date formats greatly reduces the workload of setting up and calling *strfdate*. *to_char* is such a function. *to_char* (Listing 6) accepts the address of a character array where the formatted date is to be stored, a date value, and a value which indicates the desired format of the output date string. I've defined these in the *dates.h* header file since they correspond to date formats that I use frequently. The most common date format is represented by *GREG-DATE*, which specifies the format MM/DD/CCYY, where MM is the month, DD is the day of month, and CCYY is the year. *to_char* returns the address of the character array where the date string is built. Thus, all you need to display the current date is:

```
puts( to_char( char_buffer, date(NULL), GREGDATE ) );
```

Function *to_date* (Listing 6) is the complement of *to_char*. *to_date* takes a date string plus a format indicator and calculates a date (*date_t*) value. This function is useful for conversions on date strings from data files and from the keyboard. Furthermore, since *to_date* calls *mkdate*, it can also validate the date strings.

Listing 3 Function localtime

```

/*****
/*      (c) Copyright 1993 by Stan Milam      */
/*      *****/
/*****

struct dt *localdate( date_t *arg ) {

    date_t day;
    unsigned year;
    static struct dt dt;
    int yday, mday, leapyear, month, wday;

    /*****
    /* Make sure we are in range. Upper limit is 31-Dec-9999. */
    /*****

    day = *arg;
    if ( day < (date_t) 1L || day > (date_t) MAXDATE ) return NULL;

    /*****
    /* 146097 is years_to_days(400).                      */
    /*****

    year = (unsigned) ((day * 400L) / 146097L + 1L);
    while ( years_to_days(year + 1) < day ) year++;

    /*****
    /* Compute the remaining values from what we know already. */
    /*****

    leapyear = is_it_a_leap_year(year);
    mday = (int) (day - years_to_days( year )); yday = mday - 1;
    month = days_to_months( &mday, leapyear);
    wday = (int) (day % 7L);

    /*****
    /* Assign to our internal structure.                    */
    /*****

    dt.dt_year = year;
    dt.dt_mday = mday;
    dt.dt_yday = yday;
    dt.dt_wday = wday;
    dt.dt_month = month;
    dt.dt_leap_year = leapyear;
    return &dt;
}

/* End of File */

```

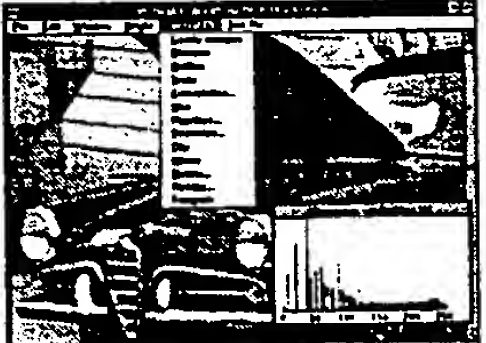
With color reduction
for fast, accurate
24-bit image display

Victor

Image Processing Library

Create Powerful Image Applications
for BMP, TIFF, PCX, GIF, TGA, and JPEG Images

- ▶ Load and save BMP/TIFF/PCX/GIF/TGA/JPEG
- ▶ Powerful grayscale and color image processing: brightness, contrast, sharpen, outline, equalize, matrix convolution, rotate, resize, and more
- ▶ Color reduction for fast and accurate display of 24-bit images
- ▶ Support for EGA/VGA/SVGA, 32K- and 16 million-color displays
- ▶ Scan b/w, grayscale, and color images with ScanJet scanners
- ▶ Print halftones, diffusion scatters, and color pictures
- ▶ Convert images between 1-, 8-, and 24-bit formats
- ▶ Convert color to grayscale
- ▶ Includes a complete image processing application with C source



Your Windows application can load and save BMP, TIFF, PCX, GIF, TGA, and JPEG files, control scanner and printer, and have powerful image processing and color reduction for the very best image display.

Victor Image Processing Library for Windows (DLL), \$295

Victor Image Processing Library for DOS, supports Microsoft and Borland C/C++ compilers, \$195

Call or fax to order
314-962-8037

Catenary Systems

470 Belleview St Louis MO 63119
314-962-7833

Victor Image Processing Library for Windows or for DOS. No royalties. Source code available. visa/mc/cod.

□ Request 253 on Reader Service Card □

Note: if the century in the date string is omitted, *to_date* uses the year value to determine the century. If the year is less than 80, *to_date* will consider the century to be 2000. If the year is greater than 79, *to_date* will consider the century to be 1900. I've included this feature to accommodate the century change which will occur in just a little over five years. Thus, to obtain a value for today's date you could use the following expression:

```
date_t date_val;
date_val = to_date( "27-Nov-93", SPELLDATE );
```

The final function in the extended suite is perhaps the most useful of all: *compute_date* (Listing 6). You can use this function to add or subtract combinations of years, months, weeks, and days to a given date value. The arguments for this function are a

Listing 4 Function mktime

```

/*****
 * (c) Copyright 1993 by Stan Milam
 */
/*****

date_t mktime( struct dt *dt ) {

    date_t rv;
    int    max_day;
    int    year, month, day_of_month, day_of_year, leap_year;

    /*****
     * Get values from the structure into local variables.
     */
    /*****

    year      = dt -> dt_year;
    month     = dt -> dt_month;
    day_of_month = dt -> dt_mday;
    day_of_year  = dt -> dt_yday;

    /*****
     * Convert the year into a sequential number, but check for
     * errors first.
     */
    /*****

    if ( year < 0 || year > 9999 ) return ( (date_t) -1 );

    rv = years_to_days(year);
    leap_year = is_it_a_leap_year(year);

    /*****
     * Try to use the month and day of month first to get the
     * number of days elapsed since the beginning of the year.
     * Use the day of the year if we do not have the above.
     * Check everything along the way.
     */
    /*****

    if ( month >= 0 && month < 12 ) {
        if ( day_of_month > 0 ) {
            rv += months_to_days( month + 1, leap_year );
            max_day = month_table[month];
            if ( month == 1 ) max_day += leap_year;
            if ( day_of_month <= max_day ) rv += day_of_month;
            else rv = (date_t) -1;
        }
        else if ( day_of_year >= 0 ) {
            max_day = leap_year ? 365 : 364;
            if ( day_of_year <= max_day ) rv += day_of_year + 1;
            else rv = (date_t) -1;
        }
        else rv = (date_t) -1;
    }
    else rv = (date_t) -1;
    return rv;
}

/* End of File */

```

Listing 5 Function strftime

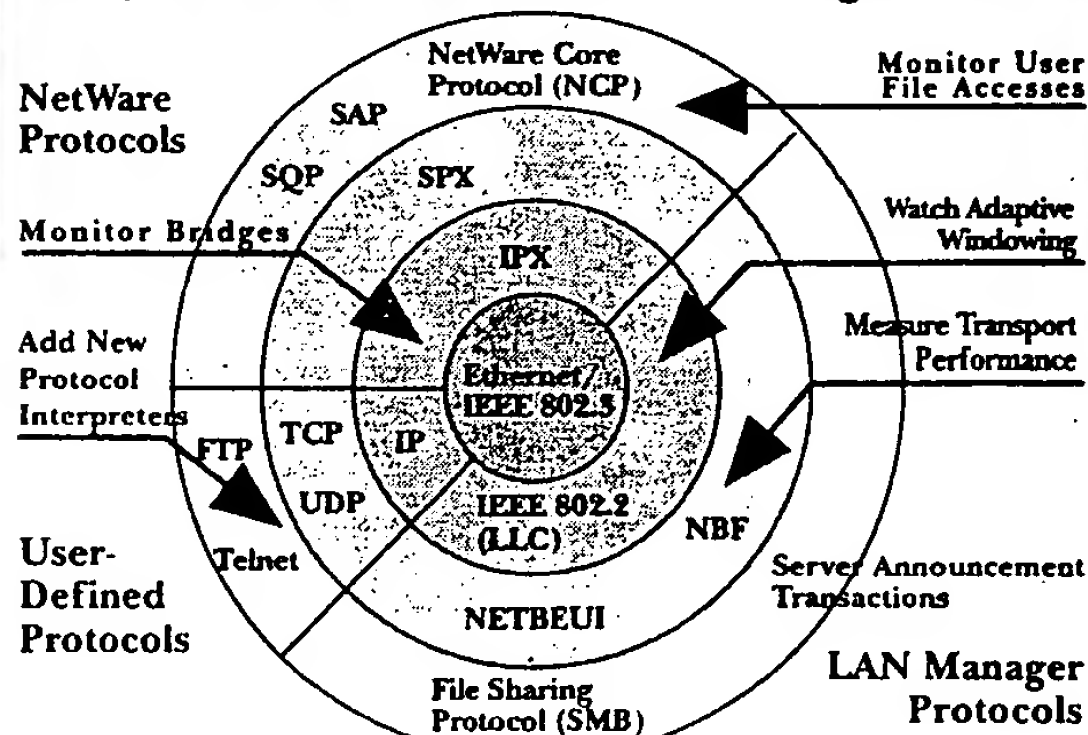
```

/*****
 * (c) Copyright 1993 by Stan Milam
 */
/*****
 * strftime()
 */
/*****
 * This is the main function in the file. It uses format char-
 * acts to format the date any way we want it in a supplied
 * memory address.
 */
/*****
 * Arguments:
 * char *buffer - Memory address where we place the formatted
 * date.
 * unsigned size - Max size of the buffer. We do not exceed.
 * char *format - The format string. Characters preceded
 * by the % character are taken to be format
 * specifiers. The format specifiers are:
 *
 * %a - Format abbreviated weekday name.
 * %A - Full weekday name.
 * %b - Format abbreviated month name.
 * %B - Format full month name.
 * %d - Format the day of the month (01-31).
 * %j - The day of the year (001 - 366).
 * %m - The month of the year (01 - 12).
 * %U - The week of year (00 - 52, Sunday)
 * %w - The day of week (0 - 6, Sunday = 0)
 * %W - The week of year (00 - 52, Monday).
 * %x - The date (MM/DD/CCYY).
 * %y - Formats a two digit year.
 * %Y - Formats the four digit year.
 *
 * struct dt *dt - A pointer to the dt structure populated
 * with the localtime() function.
 */
/*****

```

The SnooperTM Ethernet Protocol Analyzer

Become a Networking Expert with this PC-Based Analyzer for NetWare and LAN Manager LANs.



Captures, decodes into multiple layers, and displays Ethernet traffic, and lets you actually add your own protocol interpreters. With source, only \$350.

GENERAL SOFTWARE Tel. (206) 391-4285
 Fax. (206) 557-0736
 P.O. Box 2571 Redmond, WA 98073 The Protocol Experts

Copyright 1993 General Software, Inc. General Software, the GS logo, and The Snooper are trademarks of General Software. Other marks property of their owners.

Listing 5 continued

```

unsigned strfdate( char *buffer, unsigned size, char *format, struct dt *dt )
(
    size_t len;
    int start;
    unsigned rv = 0;
    char wrkbuf[15];
    char *src = format;
    char *dest = buffer;

    /******
    /* Enter a loop copying characters from the format string */
    /* into the buffer unless it is a format character. */
    /******

    while ( *src && size ) {
        if ( *src != '%' ) {
            *dest++ = *src++;
            size--; rv++;
        }
        else {
            len = 0;
            /******
            /* We have a format character so we handle it. */
            /******

            switch( *++src ) {

                /******
                /* Just in case there is nothing following! */
                /******

                case '\0' :
                    continue;

                /******

```

starting date value, plus the number of years, months, weeks, and days to add or subtract. To demonstrate the power and flexibility of this function the following sample code obtains the current date, adds one year, subtracts three months, adds two weeks, and subtracts four days. It then displays the computed date.

```

start = date(NULL);
finish = compute_date(start, 1, -3, 2, -4);
puts(to_char(buffer, finish, SPELLDATE));

```

Adding years, weeks, and days to a given date is relatively straightforward. However, date math using months is a bit complicated, because not all months contain the same number of days. For example, suppose the current date is the last day of May. If you wish to compute one month into the future, what day is it — The 30th of June, or the 1st of July? I chose to enforce the former convention, so in this case the computed date would be the 30th of June. However, if the starting date were the 30th of June and one month were added the result would be 30th of July, not the 31st. Another problem arises when adding or subtracting years when the starting date is the 29th of February in a leap year. Is the resulting date the 28th of February or the 1st of March? Again, for consistency, I chose the former convention.

Miscellaneous Date Functions

Several functions do not figure so prominently as those mentioned already, but are worth mentioning. I've already mentioned the

time_to_date function (Listing 1) in passing — it converts the values returned from *time* and *mtime* in the standard library to a date value used by the extended functions. I've implemented this conversion so as to maintain portability. The *first_day_of_month* and *last_day_of_month* functions, (Listing 6) when given a date value, determine the month in which the date value falls and returns a date value for the first day or the last day of the month respectively. The *next_day_of_week* and the *previous_day_of_week* functions (Listing 6) are useful for determining either a future or past day of the week from a given date value. For example, in our speech we often say something like "The event is three weeks from the coming Saturday." To compute this with the extended date functions you would simply write:

```

date_v =
next_day_of_week(date(NULL), SATUR-
DAY) + 21;

```

The *dates.h* header file defines values for each day of the week.

Real-Time Multitasking with DOS for Microsoft C, Borland C, Borland/Turbo Pascal

Develop Real-Time Multitasking Applications under MS-DOS with RTKernel!

RTKernel is a professional, high-performance real-time multitasking kernel. It runs under MS-DOS or in ROM and supports Microsoft C, Borland C++, Borland/Turbo Pascal, and Stony Brook Pascal. RTKernel is a library you can link to your application. It lets you run several C functions or Pascal procedures as parallel tasks. RTKernel offers the following advanced features:

- pre-emptive, event/interrupt-driven scheduling
- number of tasks only limited by available RAM
- task-switch time of approx. 6 μ s (33-MHz 486)
- performance is independent of the number of tasks
- use up to 64 priorities to control your tasks
- priorities changeable at run-time
- time-slicing can be activated
- programmable timer interrupt rate (0.1 to 55 ms)
- high-resolution timer for time measurement (1 μ s)
- activate or suspend tasks out of interrupt handlers
- programmable interrupt priorities
- semaphores, mailboxes, and message-passing
- keyboard, hard disk, and floppy disk idle times usable by other tasks
- interrupt handlers for keyboard, COM ports, and network interrupts included with source code
- supports up to 36 COM ports (DigiBoard, Hostess boards)
- supports protocols XON/XOFF, DTR/DSR, RTS/CTS
- full support of NS16550 UART chip
- supports math coprocessor and emulator
- fast, inter-network communication using Novell's IPX
- runs under MS-DOS 3.0 to 6.x, DR-DOS, LANs, or without operating system
- DOS calls from several tasks without re-entrance problems
- supports resident multi-tasking applications (TSRs)
- runs Windows or DOS Extenders as a task
- supports CodeView and Turbo Debugger
- Kernel Tracer for easy debugging
- ROMable
- full source code available
- no run-time royalties
- free technical support by phone or fax

RTKernel-C 4.0 \$495 RTKernel-Pascal 4.0 \$445
C Source Code: add \$445 Pascal Source Code: add \$375

International orders: add \$30 shipping and handling.
MasterCard, Visa, check, bank transfer accepted.

FREE DEMO DISK
Please request info kit C

On Time
MARKETING

Professional Programming Tools

In North America, please contact:

On Time Marketing
88 Christian Avenue • Selauket, NY 11733 • USA
Phone (516) 682-0034 • Fax (516) 682-1172
CompuServe 73313.317

Outside North America, please contact:

On Time Marketing
Karolinenstrasse 32 • 20357 Hamburg • GERMANY
Phone +49 40 43 74 72 • Fax +49 40 43 51 96
CompuServe 100140.633

Request 214 on Reader Service Card

Listing 5 continued

```

/* Format the abbreviated weekday name. */
/*****

case 'a' :
    len = 3;
    if ( len > size ) len = size;
    strncpy(dest, full_weekday[dt -> dt_wday],
        len );
    break;

/*****
/* Format the full name of the week. */
/*****

case 'A' :
    len = strlen( full_weekday[dt -> dt_wday] );
    if ( len > size ) len = size;
    strncpy( dest, full_weekday[dt->dt_wday], len);
    break;

/*****
/* Format the abbreviated name of the month. */
/*****

case 'b' :
    len = 3;
    if ( len > size ) len = size;
    strncpy(dest, full_month[dt -> dt_month], len);
    break;

/*****
/* Format the full name of the month. */
/*****

case 'B' :
    len = strlen(full_month[dt -> dt_month]);

```

```

    if ( len > size ) len = size;
    strncpy(dest, full_month[dt -> dt_month], len);
    break;

/*****
/* Format the day of the month. */
/*****

case 'd' :
    len = sprintf(wrkbuf, "%02d", dt -> dt_mday);
    if ( len > size ) len = size;
    strncpy(dest, wrkbuf, len);
    break;

/*****
/* Format the day of the year (001 - 366). */
/*****

case 'j' :
    len = sprintf(wrkbuf, "%03d",
        dt -> dt_yday + 1);
    if ( len > size ) len = size;
    strncpy(dest, wrkbuf, len);
    break;

/*****
/* Format the month of the year (01 - 12). */
/*****

case 'm' :
    len = sprintf(wrkbuf, "%02d",
        dt -> dt_month + 1);
    if ( len > size ) len = size;
    strncpy(dest, wrkbuf, len);
    break;

```



Add phone and
fax interfaces
to your 'C'
programs. With
the **MULTI-VOICE** and
MULTI-FAX Toolkits.

- ☎ Multi-Voice for Dialogic, Rhetorex, NewVoice, Bicom, Power Line II, VBX - \$599
- ☎ Multi-Voice for Watson board - \$99
- ☎ Multi-Fax for Intel SatisFAXtion (all models) and other CAS compatible boards - \$199
- ☎ Multi-Tasking Integrated (DESeqview version option available - \$99)
- ☎ All Library Source Code Included
- ☎ Many Example Programs



ITL Software

Phone: 514-835-3124

Fax: 514-835-4772

BBS (demos): 514-835-5945

Fax-On-Demand: 514-835-2216

STOP THE PIRATES®!
USE THE PROVEN PLUG

The proof is in our thousands of
satisfied customers - You too deserve
the best protection at the lowest price!

EliaShim's Software Protection Systems:

MEMOPLUG®

Protection based on a combination of a unique
code, a serial number and a version number.

FILES OPTION®

Memoplug-based protection for NON-executable
programs, such as AUTOCAD, LOTUS, inter-
preter data files, run time modules, macros, LISP.

LANPLUG®

Complete protection for a network with one plug.
Limits number of users.

New!
Lease your software with
CLOCKPLUG®
Call for details and availability

EliaShim
MICROCOMPUTERS INC.

Call Now: 1-800-677-1587

4005 Wedgemere Dr.
Tampa, FL 33610
TEL: (813) 744-5177 FAX: (813) 744-5197

Compilation Notes

All the date functions I've described, as well as a few internal functions are implemented in a file called *dates.c*, available on this month's code disk. For convenience, *dates.c* was split into Listings 1 thru 6 in this article. Listing 1 contains some *#includes* and declarations which appear at the beginning of *dates.c*. If you compile any of the date functions separately, be sure to include these statements in your source text.

Summary

The standard time/date functions are quite adequate for most date calculations, but their range is limited because their unit of measure is seconds. The date routines presented here overcome the limited range of the standard functions while maintaining their flexibility. I've presented three additional functions which remove much of the burden of formatting date strings, parsing date strings, and computing past or future dates. My new date

Listing 5 *continued*

```

/*****
/* Formats the week of the year. %U starts the */
/* week with Sunday, %W starts the week with a */
/* Monday.
*****/

case 'U' :
case 'W' :
    start = (*src == 'U') ? 1 : 0;
    len = sprintf(wrkbuf, "%02d",
        week_of_year(start, dt -> dt_wday,
            dt -> dt_yday));
    if ( len > size ) len = size;
    strncpy(dest, wrkbuf, len);
    break;

/*****
/* Format the day of the week (1 - 6).
*/

```

```

/*****
*****/

case 'w' :
    len = sprintf(wrkbuf, "%d", dt -> dt_wday);
    if ( len > size ) len = size;
    strncpy(dest, wrkbuf, len);
    break;

/*****
/* Formats a traditional Gregorian date with
/* single exception: The year is four digits.
*****/

case 'x' :
    len = sprintf(wrkbuf, "%02d/%02d/%04d",
        dt -> dt_month + 1,
        dt -> dt_wday,
        dt -> dt_year);
    if ( len > size ) len = size;
    strncpy(dest, wrkbuf, len);
    break;

/*****
/* Formats the year in at least four digits.
*****/

case 'Y' :
    len = sprintf(wrkbuf, "%04d", dt -> dt_year);
    if ( len > size ) len = size;
    strncpy(dest, wrkbuf, len);
    break;

/*****
/* Format the year without the century.
*****/

case 'y' :
    len = sprintf(wrkbuf, "%02d",
        dt -> dt_year % 100);
    if ( len > size ) len = size;
    strncpy(dest, wrkbuf, len);
    break;

/*****
/* User may want a % character in the output.
*****/

case '%' :
    len = 1;
    if ( len > size ) len = size;
    else *dest = '%';
    break;

) /* Switch */

src += 1;
rv += len;
dest += len;
size -= len;
) /* while */
)
if ( size ) *dest = '\0';
return rv;
)
/* End of File */

```

For software architects ...



- ☒ thinking about friendly end-user programming tools
- ☒ drafting flexible text manipulators
- ☒ organizing rapid access to huge amounts of data

... we have the building blocks

TextMatch

Table-driven text converter plus general purpose macro processor

KeyPoint

Balanced binary trees, abbreviated keyword associator, constructor for list-directed languages

DataOrgan

Sequential and keyed record set manager with B*-trees, based on time and space efficient page-oriented data storage

All in Standard C. For product profile contact:

interactive instruments

Software for Science
and Engineering

Beethovenplatz 14
53115 Bonn, Germany
Phone: +49-228-650041
Fax: +49-228-697608
CompuServe: 73064,1240

in North and South America:
European Software Connection

PO Box 1982
Lawrence, KS 66044, USA
Phone: 913-832-2070
Fax: 913-832-8787
CompuServe: 71141,3624

Credit cards accepted

Request Reader Service #337FAX #1202

ADVANCED SOLUTIONS FOR C & C++ PROGRAMMERS

C/C++ Users Journal™

Advanced Solutions for C/C++ Programmers

If you're reading this journal,
you're among a select group of
advanced programmers
worldwide who program
professionally in C and C++.

**Come meet
fellow programmers,
writers, and
C/C++ USERS JOURNAL
representatives at the**

**SD '94 EAST
October 4 - 6
Washington DC**

**See us at
Booth 530**

For more information call:

913-841-1631

We look forward to seeing you there!

Listing 6 continued

```

        if ( compare( wrkbuf, months[month], len ) == 0 )
            break;
    }
    if ( month == 12 ) return (date_t) -1;
    break;

case JULDATE :
    len = strlen( string );
    if ( len == 5 || len == 7 ) {
        len -= 3;
        sprintf(wrkbuf, "%Xdd%Xdd", len, 3);
        i = sscanf(string, wrkbuf, &year, &month, &yday);
        if ( i != 2 ) return (date_t) -1;
        yday--;
    }
    else return (date_t) -1;
    break;

case SYSDATE :
    len = strlen( string );
    if ( len == 8 || len == 6 ) {
        len -= 4;
        sprintf(wrkbuf, "%Xdd%Xdd%Xdd", len, 2, 2);
        i = sscanf(string, wrkbuf, &year, &month, &yday);
        if ( i != 3 ) return (date_t) -1;
        month--;
    }
    else return (date_t) -1;
    break;

default :
    return (date_t) -1;
}

/* If the year is a two digit year, adjust for the century */
/* change. Anything less than 80 is considered to be in */
/* 21st century. */
if ( year < 100 )
    year += year < 80 ? 2000 : 1900;

/* Setup and call mktime() to get the sequential day number */
dt.dt_year = year;
dt.dt_mday = mday;
dt.dt_yday = yday;
dt.dt_month = month;
rv = mktime( &dt );
return rv;
}

char *date_to_string(char *buffer, date_t day, int type) {
    int size;
    struct dt *dt;
    char wrkbuf[25], *format;

    wrkbuf[0] = '\0';
    size = sizeof(wrkbuf);

    switch ( type ) {
        case GREGDATE :
            format = "%X";
            break;

        case MLDATE :
            format = "%d/%m/%Y";
            break;

        case SPELLDATE :
            format = "%d-%b-%Y";
            break;

        case JULDATE :
            format = "%Y%j";
            break;
    }

```


Stan Milam

An Extended Date Library for C

functions align themselves closely with the Standard C date/time functions: Since I designed the three extended date functions to work with these new date functions, it should be easy to adapt the extended functions to work with the Standard C functions. One drawback to the date routines results from the unit of

measure being in days: The date values lose the ability to serve as timestamps. However, in many cases, the increase in range more than compensates for the loss of precision. □

Listing 6 Extended functions

```
/*
Extended Date Functions
(c) Copyright 1993 by Stan Milam
*/
```

```
date_t to_date( char *string, int type ) {
```

```
    date_t rv;
    struct dt dt = { 0, 0, 0, 0, 0, 0 };
    int len, i, mday, yday, month, year;
    char wrkbuf[16], **months = full_months;
```

```
    year = month = yday = mday = 0;
    switch ( type ) {
```

```
        case DDMCCYY :
            i = sscanf(string, "%2d%2d%4d", &mday, &month,
                &year);
            if ( i != 3 ) return (date_t) -1L;
            month--;
            break;
```

```
        case MMDDCCYY :
            i = sscanf(string, "%2d%2d%4d", &month, &mday,
                &year);
            if ( i != 3 ) return (date_t) -1L;
            month--;
            break;
```

```
        case GREGDATE :
            i = sscanf(string, "%d%*c%d%*c%d", &mday, &month, &year,
                &year);
            if ( i != 3 ) return (date_t) -1L;
            month--;
            break;
```

```
        case MILDATE :
            i = sscanf(string, "%d%*c%d%*c%d", &mday, &month, &year,
                &year);
            if ( i != 3 ) return (date_t) -1L;
            month--;
            break;
```

```
        case SPELLDATE :
            i = sscanf(string, "%d%*c%3c%*c%d", &mday, wrkbuf,
                &year);
            if ( i != 3 ) return (date_t) -1L;
            for ( month = 0, len = 3; month < 12; month++ ) {
```

Graphics & Timing Tools

BGI SVGA Video Driver Toolkit *New Toolkit!*

Provides Super VGA driver support at up to 1280x1024x256 for most popular SVGA cards using the Borland BGI graphics library. Support for graphics mode mouse and multiple video pages. Import/export popular bitmap file formats. Supports Borland DOS language compilers in real and protected mode. \$129.95 w/source.

BGI Font Toolkit *New Toolkit!*

Complete replacement for the graphics text functions in the Borland BGI graphics library. Corrects bugs in BGI text functions, allows arbitrary rotation, bolding, underlining, and italicizing of BGI fonts. Supports additional bitmap font formats for filled font rendering. Works with any BGI video or hardcopy driver. Supports Borland DOS language compilers in real and protected mode. \$89.95 w/source.

BGI Printer Driver Toolkit *New Version!*

Provides drivers for Borland's BGI graphics library to support a wide variety of printers, plotters, and bitmap file formats. Support for EMS/XMS. Print popular bitmap file formats. Extensive color device support. Not a screen dump - load our drivers with BGI's *initgraph* and get full output device resolution. Supports Borland DOS language compilers in real and protected mode. \$129.95 w/source.

BGI For Windows *Port DOS BGI Code to Windows*

Gives you an interface to the Windows 3.x GDI compatible with Borland BGI graphics calls. Port your DOS BGI graphics code effortlessly to Windows. Full support for 256 color palettes, BGI stroke fonts, high resolution displays, and rasterized hardcopy. BGI extensions support TrueType and 24 bit color. Supports Borland Windows language compilers. \$129.95 w/source.

PC Timer Tools *Event Timing and Scheduling*

Brings microsecond resolution timing to your DOS application with extensive functions for timers, delays, alarms, timer tick management, and thread scheduling. Ideal for execution profiling, data acquisition, and process control. Supports Borland DOS compilers, MSC/C++, Intel 386 CB, Zortech, \$89.95 w/source. PC Timer Objects for C++ and Turbo Pascal OOP - \$89.95 w/source.

All toolkits include *The Official Fine Print*

complete source (in both C and Turbo Pascal), object/driver/DLL distribution license, and our 30 day "No Questions Asked" return policy. Add \$5 shipping USA, \$10 elsewhere. VISA, MasterCard, and American Express accepted. Our toolkit code is found in a wide variety of commercial and private applications in daily use by over 100,000 end users

Ryle Design

Purveyors of Big Science since 1987

PO Box 22, Mt. Pleasant, MI 48804 USA

Voice/Fax: 517.773.0587 CIS: 73047,1765

Demos and spec sheets available on our BBS: 517.772.2393

← EditMaster™2 →

"The Ultimate BRIEF® Upgrade is Even Better!"

...from the BRIEF add-on professionals at Software Annex.

Bring BRIEF up-to-date with Version 2 of our comprehensive upgrade package, professionally crafted and fully tested.

ASCII/ANSI Table ~ Improved Buffer List
Count chars, words, lines ~ Line & Box Drawing
Matching Brace Location ~ File Difference Location
Next/Previous Function Finder ~ Grep (buffers or files)
Glossary/Template Expansion ~ Jump to Column
Subdirectory Recursion in File Prompts
Named Scrap ~ Full Scrap History
Search for Marked Block ~ Show Assigned Keys
Show Unassigned Keys ~ Show size of Marked Block
Sum Numbers ~ Synchronized Window Scroll
Modification date Update ~ Better Bookmarks
File Overwrite Protection ~ Write before Compile
No-prompt Window Operations ~ Enhanced Exit
Enhanced Search and Translate
Wildcards and File lists in File Prompts
Easily save Grep, Scrap, and Cmd History in STATE file
Multiple Keystroke Macros (Permanent or Temporary)
Run System Commands & Capture Output
Updated 70+ page Manual ~ For both DOS and OS/2

(Highlighted features new or improved in Version 2)

...still only **\$89⁹⁵** includes SOURCE!

(Upgrade: \$24.95)

European Orders,
Call:


GREY MATTER

Tel: +44 (0)364 654100

Fax: +44 (0)364 654200

Order Toll-free

1-800-453-5277

For technical
info contact:
 10210 W. 26th Ave
Suite 7
Lakewood, CO 80215
USA
Tel: 303.274.8088
Fax: 303.274.8279

Software
ANNEX

Requires BRIEF 3.0 or later, runs under DOS or OS/2. BRIEF is a registered trademark of Borland International, Inc.

□ Request 261 on Reader Service Card □

□ Request 279 on Reader Service Card □

Listing 6 *continued*

```

case SYSDATE :
    format = "%Y%m%d";
    break;

case DDMMCCYY :
    format = "%d%m%Y";
    break;

case MMDDCCYY :
    format = "%m%d%Y";
    break;

default :
    format = NULL;
}

if ( format != NULL ) {
    if (( dt = localtime( &day )) != NULL )
        strftime(wrdbuf, size, format, dt);
}
return (strcpy(buffer, wrdbuf));
}

char *to_char(char *buffer, date_t day, int type) {
    return date_to_string(buffer, day, type);
}

date_t next_day_of_week( date_t value, int day ) {
    date_t rv;
    struct dt *dt;

    /* See if we have a valid date value. */

    if ( (dt = localtime( &value )) == NULL )
        rv = (date_t) -1L;
    else if ( day < SUNDAY || day > SATURDAY )
        rv = (date_t) -1L;
    else {
        /* Compute how many days until the next specified day of
        /* the week. Make sure we are within our limits too. */
        rv = (date_t) 7 - dt->dt_wday + day;
        if ( rv > (date_t) 7 ) rv -= (date_t) 7;
        rv = rv + value > MAXDATE ? (date_t) -1L : rv + value;
    }

    return rv;
}

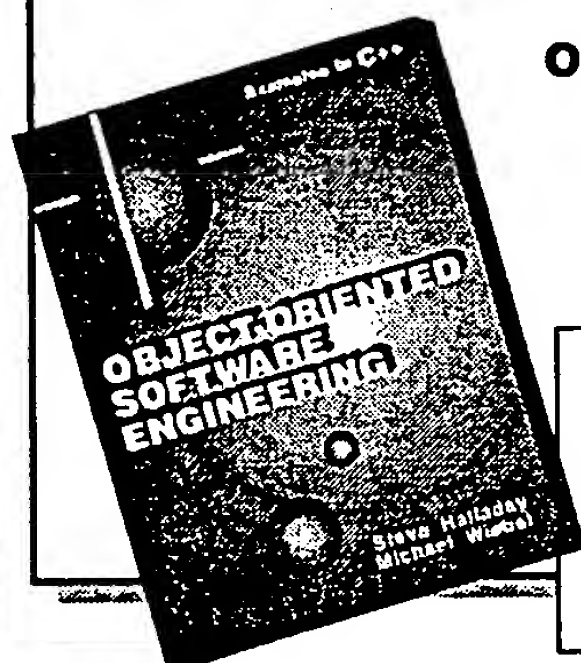
date_t previous_day_of_week( date_t value, int day ) {
    date_t rv;
    struct dt *dt;

    /* Check for a valid date value. */

    if ((dt = localtime( &value )) == NULL)
        rv = (date_t) -1L;
    else if ( day < SUNDAY || day > SATURDAY )
        rv = (date_t) -1L;
    else {
        /* Compute the date value for previous specified day
        /* and do sanity check. */

```

IMPROVE YOUR SOFTWARE ENGINEERING WITH OBJECT-ORIENTED METHODS



OBJECT-ORIENTED SOFTWARE ENGINEERING

Learn how to:

- Engineer applications to minimize costs
- Manage all six phases of the software lifecycle
- Use recursion in the object-creation process

USE RECURSION FOR PRECISE DESIGN MANAGEMENT

This book demonstrates object-oriented principles for each phase of development, from specification through maintenance. A comprehensive example coded in C++ ties the demonstration together. For precision, the design method is presented in a pseudocode recursive algorithm.

**ONLY
\$29.95**
Plus Shipping

ORDER TODAY!

Order: Book T37 *Object-Oriented
Software Engineering*

913-841-1631

FAX 913-841-2624



Real-time Multi-tasking Full C source Code Included

The Tics Realtime multi-tasking kernel is a linkable C library that allows C functions to run as preemptive concurrent tasks. Developed for professional software developers who need a capable full-featured real-time multi-tasking development system. Tics can run with MS-DOS or stand-alone on an embedded target. No royalties. Tics also includes a multi-tasking COM port library for multi-tasking serial communications.

- C source code is included - fully documented and complete.
- Includes our new book "The Art of Real-time Programming".
- Tics is written entirely in C and is portable to virtually any microprocessor.
- Includes integrated COM port library for multi-tasking RS-232 applications.
- The PC timer chip is reprogrammed to user specified granularity.
- Preemptive, cooperative, or time-sliced scheduling, configurable on a task by task basis.
- No restriction on the number of tasks.
- Stack size is variable on a task by task basis.
- Priorities can be changed dynamically.
- Tasks may be created dynamically.
- The development system also includes Tiny Tics, a small round robin kernel that is ideal for micro-controllers.
- Tics requires less than 8K bytes of code space and is ROMable.
- Three types of high speed timers: in-line pause, one-shot timers, and time critical periodic timers.
- Differential timer management system allows for an unrestricted number of concurrent timers without increasing time spent in the timer isr.
- Inter-task communication using message queues or high speed mailboxes.
- Critical region management.
- High speed memory management system. Unrestricted number of memory pools.
- Priorities on tasks and messages provide great flexibility. Priorities on messages means that high priority messages go straight to the front of the queue.
- Optionally, cooperative tasks may share a common stack so that hundreds of tasks can run with the overhead of a single stack.
- Object oriented. Create task instances with their own separate instance data. Can run with C++ for real-time OOP.
- Messages or mail can be sent from within an isr.
- Time out option while waiting for a message.
- Includes manual and source code for sample applications that include dials, gauges, data acquisition, display, and more.



1584 Camden Village Circle
San Jose, CA 95124
(408) 723-2200

Price: \$499

Listing 6 *continued*

```

    rv = (date_t) day - dt->dt_wday;
    if ( rv >= (date_t) 0 ) rv -= (date_t) 7L;
    rv += value;
    if ( rv < (date_t) 1L ) rv = (date_t) -1L;
}

```

```

    return rv;
}

```

```

date_t last_day_of_month( date_t value ) {

```

```

    date_t rv;
    struct dt *dt = localtime( &value );

```

```

    /* See if date value was valid. */
    /* ***** */

```

```

    if ( dt == NULL )
        rv = (date_t) -1L;

```

```

    else {
        /* Determine last day of the month. compute the difference and add to the date value. */
        /* ***** */
        rv = month_table[ dt->dt_month ];
        if ( dt->dt_month == 1 ) rv += dt->dt_leap_year;
        rv = rv - dt->dt_mday + value;
    }

```

```

    return rv;
}

```

```

date_t first_day_of_month( date_t value ) {

```

```

    date_t rv;
    struct dt *dt = localtime( &value );

```

```

    rv = ( dt == NULL ) ? -1L : (date_t) 1 - dt->dt_mday + value;
    return rv;
}

```

```

date_t compute_date( date_t value, int years, int months, int weeks, int days ) {

```

```

    date_t rv; /* Return value */
    int ld1, ld2; /* Last days of months */
    struct dt *wrk; /* A date structure */

```

```

    wrk = localtime( &value ); /* Get a date structure */
    if ( wrk == NULL ) return (date_t) -1L;

```

```

    /* Go ahead and compute the weeks and days. */
    /* ***** */

```

```

    rv = (weeks * 7) + days;

```

```

    /* Now compute the months. We may have to adjust the years */
    /* value. */
    /* ***** */

```

```

    if ( months ) {
        if ( abs(months) > 11 ) {
            years += months / 12;
            months = months % 12;
        }
    }

```

```

    /* Now compute the target month and handle wrap-around. */
    /* ***** */

```

```

    months += wrk->dt_month;
    if ( months < 0 ) {
        months += 12;
        years -= 1;
    }

```

```

    else if ( months > 11 ) {
        months -= 12;
        years += 1;
    }

```

```

    /* Now we have to make adjustment if we are on the last day of the month. For instance if date is 31 May and we add 1 month the result should be 30 June. */
    /* ***** */

```

```

    ld1 = month_table[wrk->dt_month];
    if ( wrk->dt_month == 1 ) ld1 += wrk->dt_leap_year;
    if ( wrk->dt_mday == ld1 ) {
        ld2 = month_table[months];
        if ( months == 1 ) ld2 += is_it_a_leap_year(years + wrk->dt_year);
        if ( ld1 > ld2 ) wrk->dt_mday = ld2;
    }
    wrk->dt_month = months;
}

```

```

    wrk->dt_year += years;

```

```

    /* The following is a fixup in case we started on Feb 29 and we are computing into another year. */
    /* ***** */

```

```

    ld1 = month_table[wrk->dt_month];
    if ( wrk->dt_month == 1 ) ld1 += is_it_a_leap_year(wrk->dt_year);
    if ( wrk->dt_mday > ld1 ) wrk->dt_mday = ld1;

```

```

    /* Call mktime() to do the work. */
    /* ***** */

```

```

    rv += mktime( wrk );
    return rv;
}

```

```

/* End of File */

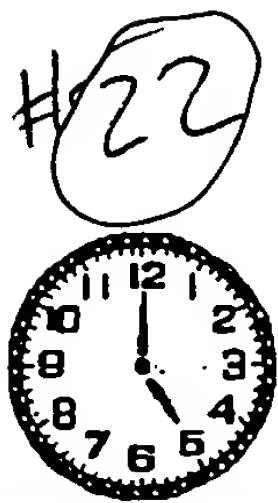
```

BASIC to C++

B2CPP translates BASIC source code to C++ source code. But this is just a beginning. The translated C++ source code is structured even if the original BASIC source code is spaghetti code. Variables are scoped to local or global depending on their usage. Dead code and variable are eliminated. String and array class, overloaded function increases readability. Many options for different needs. **Free demo disk is available.**

Gotoless Conversion

7105 Dee Cole Drive
The Colony, Texas 75056, U.S.A.
(800) 617-2323 Voice
(214) 625-2323 Voice
(214) 370-2612 Fax/BBS



TICK, TICK, TICK...

A forum for computer people concerned about the year 2000.

VOLUME 1 NO 1

WINTER 1993

TEMPUS FUGIT...

Today is the first day of the rest of the millennium. As we go to press (or copy machine, if you prefer) there are only seven years and a few days remaining until the first day of the next one.

Hello, and welcome to the first issue of TICK, TICK, TICK, It is meant to be a forum for people in the Information Services industry concerned about the impact of the year 2000 on the Information Services industry. It will be distributed free of charge to everyone who contributes material we can use.

These pages are open to all who have something to share, and that includes vendors. As a general policy, we will not identify contributors unless they have allowed us to do so or they are vendors. If you want to get in touch with the author or subject of an article appearing here, just give us a call.

At this point most of you are probably asking yourselves why you've gotten this issue. After all you didn't send in anything we could use, so you're not entitled to it. And you'd be right. But because you've let us know you're

concerned about the millennium's effect on computers that makes you too valuable to just slough off.

Perhaps you didn't send anything because you've been too busy. Or maybe you weren't sure what we wanted. Or it might be the muse hasn't struck yet. Whatever your reason, this issue is on us. If you like it, maybe you'll submit something so you can keep getting it. But if you're still suffering from "writer's block," we'll give you yet another chance. The last two pages of this issue contain a questionnaire. If you fill it out and mail it to us, we'll send you the next issue. After that, we're not sure, but we'll come up with something.

To make everything clear and to relieve some anxieties as to who is officially a subscriber we have listed below the names of all those who hold subscriptions.

SUBSCRIBERS: Willy Wilson, John Leatherman, Robert Wachtel, David Brask, Bill Woolfolk, Bob Hartman-Berrier, Brian Pitts, Richard Langston, Michael Murray, Gordon Parnell, Janet Butler, Lech Lesiak, Mickey Williamson and Lilian Woolmer.

The reason we don't have paid subscriptions is that we don't want to do all the book-

keeping and the renewal letters, etc. Anyway we don't intend to do all the work ourselves. We like to think of TICK, TICK, TICK... as a clearinghouse for other people's material. Since we can only lure your submissions with our subscriptions, it doesn't seem fair to give them away to those who haven't contributed.

We have received some amusing and chatty letters but unfortunately some of them had nothing we felt would add to our readers knowledge. Putting the kibosh on someone's efforts does not appeal to us, and we consider it an unpleasant task. Nevertheless, as editors our first duty is to our readers.

So please don't send in any more stuff explaining the history of the Julian or Gregorian calendars. That's a dead issue. Also, please do not lament the lack of foresight on the part of our predecessors (many of whom are still around and might even be readers). We all know what went wrong.

TABLE TROUBLES

Although we are concerned chiefly with the year 2000, we want to hear about other kinds of date problems as well, e.g. leap year, end-of-decade, leap seconds etc. We are well aware of the Big Problem facing two-digit year fields, but we're just finding out about the newer ones that are popping up in the latest software. For instance, in the February 3, 1992 issue of COMPUTERWORLD there is

an item about IBM's release 4.2 of the RESOURCE MANAGEMENT FACILITY which prints statistical reports of computer use. They apparently switched the days for February and January so that January got 29 and February got 31. As of January 30 all reports were off by two days. While not critical, this is the sort of thing that could add to the confusion when we come to that big date change.

We once found out about an end-of-decade problem in the Paradox User's Journal. And the following comes from the November 17, 1992 edition of the Wall Street Journal:

TANDEM TANTRUM

"A glitch that suddenly appeared around the world in a line of supposedly fail-safe machines from **Tandem Computer Inc.** forced the company to recommend an embarrassing remedy: Unplug the machines."

The bug appeared in New Zealand at 3 PM on November 1, when the computer suddenly told its operators that the current date was December 1, 1983. It then worked its way west knocking out a cellular phone company's computer in Stockholm for nine hours, and causing a Harrisburg, Pa. newspaper to print only one of its three editions for the day. The problem was faulty software, and the company said it only affected its five-year-old NonStop CLX mini-computers, of which they had sold about 2500. A newer version that had been introduced in 1991 was not affected.

Tandem expects to have another version of the software ready in 1993 that will resolve the matter. If the users do not bother to install it, then sometime in 2001 the computer will decide the date is November 1, 1992, which is when this happened the first time.

We like items like this which show the fragility of date routines around the world. So if you see one and send it in, there's a subscription waiting with your name on it.

HERE'S ONE ALTERNATIVE DATE METHOD...

In 1989, we attended the annual meeting of the Software Maintenance Association in Atlanta. An Information Systems executive from Coca-Cola was the keynote speaker, and during the question period we asked him what he was going to do about the year 2000. He replied that he hoped to be retired. In fairness, we were speaking from the back of the room and he probably didn't hear our question clearly. When someone up front clarified it for him, he admitted they hadn't done much thinking about it. We mention this incident because afterwards Rick Langston of the SAS Institute told us he was doing something about the year 2000, and was glad to meet someone else interested in the problem. His was the first reply to our request for articles, so we are pleased to let it lead off.

"I have been responsible for maintaining the date-time algorithms for our software,

the SAS System, which is a large-scale Information Delivery System used in over 10,000 sites around the world. Within the user's programming language aspects of our system, we provide a wide range of date, time, and date-time 'informats' to read in data information, and 'formats' to write out different representations. We use a base date of January 1, 1960, and use a date count (positive or negative) from this base for our 'SAS Date value' and a second count (positive or negative) for our 'SAS Datetime value.' This method helps to avoid most problems concerning the transition to the 21st century. However, our users can read in character strings in MMDDYY or YYMMDD representations in which only two-digit years are seen. In order to properly determine the century for a two-digit year, we provide an option called YEARCUTOFF. This option is set to the year that is the first of a 100-year cycle to use in determining the proper year to match with a two-digit year. For example, if YEARCUTOFF= 1980, this means that the two-digit years of 80 through 99 will correspond to 1980 through 1999, while the two-digit years 00 through 79 will correspond to 2000 through 2079. Our current default value for YEARCUTOFF is 1900, but we will probably change this to a value like 1910 with our next major release. Of course, we recommend that our users begin changing their data files and programs now to avoid two digit years that need to be input into programs."

... AND HERE'S ANOTHER

In a similar vein, a letter to the editor of COMPUTER-WORLD speaks of the Pick operating system which uses a date format based on the days from a starting point of Jan. 1, 1968. Feb. 1, 1991 would be 8433, and Jan. 1, 2000 will be 11689.

We know there are other methods for pseudo dates and we'd like to hear about them.

A PROPER PROPOSAL

Last August a man from a manufacturing company called us to say he had been put in charge of preparing their corporate systems for the year 2000 and he was looking for help, particularly software tools. He and his team had been working on this project since last February, meeting for two hours every week. Their conversion task seemed awesome because the company has 1300 programs and 150 databases (although some of them had only a date record). He had estimated it would take 30 man-years to complete, and figured the cost at \$1,500,000. He had called other companies to find out what they were doing and was dismayed to find out they were doing almost nothing. He said that most of the corporations he had spoken to had pushed the task down to the division level and the people there seemed lost.

We called him back in November to see if he had made the October deadline for submitting his recommendations to management. He replied that he had, and he was happy to share

them with us:

1. Establish standards for date storage and usage.
2. Get a COBOL compiler with a four-digit year. They cannot enhance their current compiler but IBM has two with four-digit years.
3. Use a data analysis or reengineering tool to reduce profusion of data names. This would reduce the time and scope of effort, and improve the quality of effort. (They did not recommend a data dictionary for company reasons).
4. Develop date conversion modules for interim support for CCYY. (CC stands for the century - ed.). This will limit the extent of cross-system changes until that effort is justified on mainframe applications.
5. Use the Life Cycle approach to install full support for CCYY.
6. Examine systems, identify when they will "go critical", prioritize and set up a timetable.

And above all, once the plans have been set, follow through.

He also included a copy of a page from their Policy & Procedures manual pertaining to Internal Control Formats. It states that dates may be in alpha or numeric format, but "If the numeric format is used, it **MUST** (emphasis his - ed.) be stored as an unsigned numeric field. The use of packed or signed zone decimal data formats is **not permitted**. This is because of the increased use of distributed processing, which in turn, requires that ASCII data representation be used." He pointed out that downloading

from mainframe to PC cannot handle EBCDIC's COMP-3 (PACKED) or signed zoned format.

...8, 9, 10, THEN WHAT?

A letter from a respondent in New Jersey included a copy of an article from The New York Times about the Gregorian calendar. An interesting article, but what really piqued our interest was his remark, "Where I am now has problems when the decade changes, never mind the century." We sent him a note asking him to elaborate on his problem.

ONE MAN'S OPINIONS

The writer from a one-man shop in Ohio says he is in good shape because he doesn't have that much source code and the "system is fairly well documented." He tells us that IBM has addressed the problem of the four-digit year in the AS/400, and that they have apparently come out with a value added software package for the S/36 which will take care of it there also."

He proposes two solutions for the rest of us: "Probably the best long-term solution is to replace all occurrences of a two-digit year with four digits"; and "Allow the existence of a 2-digit year in the database, but modify it by subroutine on file I/O to a 4-digit program field - 50 thru 99 are twentieth century, 00 thru 49 are twenty-first. Granted this only works up to 2049, but (i) we'll all have new computer systems in fifty years, and (ii) I'll be dead by then..."

So there you have it, the yin and the yang of millennium maintenance. The thoroughly professional approach versus something bordering on the "quick and dirty".

It's our impression that many people are currently looking to the simpler solution for their salvation, but we feel most will come around to the more formal approach after they have investigated further and gotten an eyeful of all those worms. We'd like to get your opinion on this.

THIS MAN WANTS IDEAS...

Brian Pitts is the Project Manager of Systems Support for the Berry Company in Dayton, Ohio. His group is responsible for the maintenance and support of some of their oldest and most used application systems. They have been kicking around ideas such as system rewrites and conversions to 4-digit years, but have made no plans to start on the process. He would like to exchange ideas with others in a similar situation. His phone is 513 296-4899, and his FAX is 513 296-2259.

...AND THIS ONE WANTS TOOLS

We have received a letter from a medical care company saying they have started the ball rolling by retaining an outside consultant to assist them in the final planning phases of their Date and Medical Record Number Expansion Project. They have 15 date processing subprograms which perform some rather esoteric date calculations. In addition to convert-

ing Julian to Gregorian (and vice-versa), they check to see if a date is on a holiday calendar, calculate the work days between two Gregorian dates excluding holidays and weekends etc. All of their programs require and produce year data containing 4 digits. Right now, they are keenly interested in "the identification and feasibility of any software tool(s) which can help isolate lines of code (primary, secondary and tertiary) that will require modification."

We don't think we should identify the company or individual for two reasons: We don't think he wants to be assailed by salesmen, and we want to know what's going on out there too. So if you have a tool to promote, help us share it with everyone.

SOME SIMPLE SUGGESTIONS

Bob Wachtel, a maintenance consultant from Occidental, California sends along these suggestions: "I have made this rather simple practical suggestion to clients and colleagues: Each time a program is examined during maintenance and some potential Year-2000 problem is encountered, save the information in a small database. For example, Data-Name, Line Number pairs would identify such problem areas. Later, through use of a cross reference, other statements using that Data-Name can be easily located. This is not a definitive method, but such a database can provide concrete information on the scope and severity of the Year-2000 problem for a given program.

"I've also recommended the use of what I call an Analysis Database for maintenance. This associates a Change Number, Sample Test Conditions and Cases, and Program Location where the change applies. This could be easily extended to include Year-2000 problem information as described above."

FORESIGHT BEATS HINDSIGHT

David Brask from Brask Technologies out in Illinois is feeling right proud of himself. He writes:

"Starting in 1983 I adopted year 2000 optimism: I assumed every program would still be in use by the year 2000. All programs since that time have had 8 digit dates. Formatting is nn/nn/nn but internally the 19 is carried for comparisons. Currently the date routine used forces a 19 and will need to be upgraded to choose 19 or 20. Primary software is now in 500 different companies worldwide so I am glad I planned ahead to avoid the need for a conversion."

IN OUR NEXT ISSUE: We'll review a software tool called the General Purpose System Analyzer(GPSA); tell you what some Canadians have been doing to prepare themselves for the Big Day; and report the results of our survey.

TICK, TICK, TICK...
VOL 1, NO.1 WINTER 1993
2 GRACE COURT 2-U
BROOKLYN, NY 11201
718 858-6890 FAX (CALL FIRST)

©COPYRIGHT 1993

EDITOR/PUBLISHER - BILL GOODWIN
CHIEF COPY EDITOR - ROSANNE DOBBIN

SURVEY

THIS IS TO RESEARCH THE EXTENT OF PREPARATIONS FOR THE YEAR 2000 AMONG THE RESPONDENT SHOPS. THE RESULTS WILL BE PUBLISHED IN THE NEXT ISSUE OF TICK, TICK, TICK.... NO COMPANY OR INDIVIDUAL WILL BE IDENTIFIED IN THE REPORT. NO SALESMAN WILL SEE OR GET THIS INFORMATION.

NAME: _____

TITLE: _____

INDUSTRY: _____

COMPANY: _____

HARDWARE: _____

ADDRESS: _____

OPERATING SYSTEM: _____

PHONE: _____

. HAS YOUR SHOP STARTED TO WORK ON THE YEAR 2000? _____

. IF YES, WHEN DID YOU START? _____

. IF NO, WHEN WILL YOU START? _____

. HOW MANY BATCH PROGRAMS WILL YOU HAVE TO MODIFY? _____

DATABASES? _____ ONLINE PROGRAMS? _____

. HOW MANY PROGRAMMERS AND ANALYSTS ON STAFF? _____

. HOW MANY PEOPLE-YEARS (used to be MAN-YEARS) DO YOU THINK IT WILL REQUIRE TO FIX EVERYTHING? _____

. HOW DO YOU INTEND TO STAFF THE PROJECT? _____

. WHAT DO YOU THINK WILL BE THE EASIEST PART OF THE PROJECT?

9. WHAT DO YOU THINK WILL BE THE MOST DIFFICULT PART? _____

10. WHEN DO YOU EXPECT TO BE FINISHED? _____

11. OF THE APPLICATION PACKAGES YOU ARE NOW RUNNING, WHICH WILL NOT HANDLE 1/1/2000? _____

12. DO YOU INTEND TO REPLACE THESE PACKAGES, MODIFY THEM YOURSELF OR HAVE THE VENDOR FIX THEM? _____

13. CAN YOU DESCRIBE YOUR APPROACH TO THE PROBLEM OF THE TWO-DIGIT YEAR? _____

14. DO YOU SEE ANY PROBLEMS UNIQUE TO YOUR SHOP OR INDUSTRY? _____

15. DO YOU EXPECT TO PURCHASE PROGRAMMING TOOLS SPECIFICALLY TO HELP YOU WITH THIS PROBLEM? _____ WHAT KIND? _____

16. WHAT DO YOU THINK YOUR BUDGET WILL BE? _____

17. ANYTHING YOU WOULD LIKE TO ADD _____

PLEASE MAIL THIS TO:

TICK, TICK, TICK...
2 GRACE COURT 2-U
BROOKLYN, NY 11201



IT'S NOT WHETHER YOU WIN OR LOSE, ✱



[IPN Home](#) | [Search](#) | [Order](#) | [Shopping Cart](#) | [Login](#) | [Help](#)

May 1986 ... IBM TECHNICAL DISCLOSURE BULLETIN ... <-- -->

Title: **Date Adjustment At Turn of Century.**

Index terms: **Programming**

Text



The date is a frequently used item in data processing. The date is typically stored in a format where the year is a two-digit number with the high order digits truncated since they have always been "19". When the year 2000 arrives, havoc is threatened.

Normally, the date is compared internally using a YYMMDD or YYDDD format. When comparing dates of like format, the date which generates a "high" condition code setting is considered the most recent date. After 1999 that will not necessarily be true.

Every component which ever uses the date for purposes of deciding a "most recent" situation, must use a simple adjustment factor. That factor would be fixed at each IPL. would be a two-digit unsigned number...AA - YY where YY is the low order two digits of the current year and AA is an adjustment value with a default of 50. A negative result must be complemented. Using 50 makes 50 years in the past least recent and 49 years in the future most recent; assuming that all of the years fit within the 100 year span.

In 1984, the adjustment would be $50 - YY = -33$, or 67 complemented. Comparing 1983 vs 1984 would be $(83 + 67)$ vs $(84 + 67)$ or 50 vs 51. 51 is high and, therefore, 1984 is most recent.

If the above were modified using 10 to compute adjustment factor, the comparison result would be the same with $10 - YY = -73$ or 27, when complemented, being used for adjustment.

The example which best demonstrates the need for the adjustment is at the onset of the year 2000. Assume that the computer was IPL'd on the last day of 1999 and that there are two dates which may be only minutes apart in their creation, with one before midnight and the other afterward. The normal date comparison would result in December 31, 1999 being regarded as the most recent date for deciding which resource to use. Assume the dates are stored as (YYMMDD) 991231 and 000101. It is obvious that January 1, 2000 is most recent, but the old comparison implementation doesn't work.

Since the year is hypothetically now 2000 and 1999 was entered at the last IPL, the adjustment factor is 51; using $50 - 00$ (1999) = -49 complemented. Year 00 becomes 51 and year 99 becomes 50 so that the comparison shows high for actual year xx00; therefore, it is most recent.

What the adjustment number really means is how many years back or forward to go with valid years. The number 10, which produces the adjustment factor 27 in 1983 gives an effective range of years from 1973 to 2072. For all years except the current and the next year to be in the past, 98 is used. Using 98 in 1983, the adjustment factor would be 15. Thus, 1983 would adjust to 98 and 1984 becomes 99 - the range being 1885 to 1984.

It may be important to note that only the working versions of the dates, at the time that they are being compared are ever modified. They would continue to be stored and displayed in the familiar formats.

Diagrams: none

Order/Fcode/Docket:

86A 61471 / 75-000 P200 / RO8830297

:

May 1986

... IBM TECHNICAL DISCLOSURE BULLETIN ...

<= =>

[Privacy](#) | [Legal](#) | [IBM](#) | [FAQ](#) | [Feedback](#) | [Contact Us](#)

Date Adjustment At Turn of Century.

#26

The date is a frequently used item in data processing. The date is typically stored in a format where the year is a two-digit number with the high order digits truncated since they have always been "19". When the year 2000 arrives, havoc is threatened.

Normally, the date is compared internally using a YYMMDD or YYDDD format. When comparing dates of like format, the date which generates a "high" condition code setting is considered the most recent date. After 1999 that will not necessarily be true.

Every component which ever uses the date for purposes of deciding a "most recent" situation, must use a simple adjustment factor. That factor would be fixed at each IPL. would be a two-digit unsigned number...AA - YY where YY is the low order two digits of the current year and AA is an adjustment value with a default of 50. A negative result must be complemented. Using 50 makes 50 years in the past least recent and 49 years in the future most recent; assuming that all of the years fit within the 100 year span.

In 1984, the adjustment would be $50 - YY = -33$, or 67 complemented. Comparing 1983 vs 1984 would be $(83 + 67)$ vs $(84 + 67)$ or 50 vs 51. 51 is high and, therefore, 1984 is most recent.

If the above were modified using 10 to compute adjustment factor, the comparison result would be the same with $10 - YY = -73$ or 27, when complemented, being used for adjustment.

The example which best demonstrates the need for the adjustment is at the onset of the year 2000. Assume that the computer was IPL'd on the last day of 1999 and that there are two dates which may be only minutes apart in their creation, with one before midnight and the other afterward. The normal date comparison would result in December 31, 1999 being regarded as the most recent date for deciding which resource to use. Assume the dates are stored as (YYMMDD) 991231 and 000101. It is obvious that January 1, 2000 is most recent, but the old comparison implementation doesn't work.

Since the year is hypothetically now 2000 and 1999 was entered at the last IPL, the adjustment factor is 51; using $50 - 00$ (1999) = -49 complemented. Year 00 becomes 51 and year 99 becomes 50; so that the comparison shows high for actual year xx00; therefore, it is most recent.

What the adjustment number really means is how many years back or forward to go with valid years. The number 10, which produces the adjustment factor 27 in 1983 gives an effective range of years from 1973 to 2072. For all years except the current and the next year to be in the past, 98 is used. Using 98 in 1983, the adjustment factor would be 15. Thus, 1983 would adjust to 98 and 1984 becomes 99 - the range being 1885 to 1984.

It may be important to note that only the working versions of the dates, at the time that they are being compared are ever modified. They would continue to be stored and displayed in the familiar formats



Vendors

Search

31

DOOMSDAY 2000

by Peter de Jager

pdejager@year2000.com

ComputerWorld, September 6, 1993

The date change in the year 2000 - an event that may trigger fatal errors in mission-critical systems - is only 2,308 days away. Many IS people are unprepared or unconcerned.

Have you ever been in a car accident? Time seems to slow down as you realize you're going to crash into the car ahead of you.

It's too late to avoid it - you're going to crash. All you can do now is watch it happen.

The information systems community is heading toward an event more devastating than a car crash. We are heading toward the year 2000. We are heading toward a failure of our standard date format: MM/DD/YY.

Unfortunately, unlike the car crash, time will not slow down for us. If anything, we're accelerating toward disaster.

This is a good news/bad news story. First the bad news: There is very little good news. There is no way to avoid the fact that our information systems are based on a faulty standard that will cost the worldwide computer community billions of dollars in programming effort.

Perhaps more importantly, we are going to suffer a credibility crisis. We and our computers were supposed to make life easier; this was our promise. What we have delivered is a catastrophe.

The problem is twofold: the date issue itself and, more importantly, our reluctance to address the problem.

ProblemID

What exactly is the "problem"? To save storage space - and perhaps reduce the amount of keystrokes necessary to enter a year - most IS groups have allocated two digits to the year. For example, "1993" is stored as "93" in our data files, and "2000" will be stored as "00". These two-digit dates exist on millions of data files used as input to millions of applications.

This two-digit date affects data manipulation, primarily subtractions and comparisons. For instance, I was born in 1955. If I ask the computer to calculate how old I am today, it subtracts 55 from 93 and announces that I'm 38.

So far so good. But what happens in the year 2000? The computer will subtract 55 from 00 and will state that I am -55 years old. This error will affect any calculation that produces or uses time spans, such as an interest calculation.

If you have some data records and want to sort them by date (e.g., 1965, 1905, 1966), the resulting sequence would be 1905, 1965, 1966. However, if you add in a date record such as 2015, the computer, which reads only the last two digits of the date, sees 05, 15, 65, 66, and sorts them incorrectly.

These are just two types of calculations that are going to produce garbage. There are others.

The task facing us is to identify and correct all the date data and check the integrity of all calculations involving the date information. We must correct the data residing in all data files or write code to handle the problem.

The starting point

How do we identify the problem data and the associated calculations? We have few, if any, standards for labeling data used in date calculations. The only choice we have is to examine each line of code and make the necessary changes.

One IS person I know of performed an internal survey and came up with the following results: of 104 systems, 18 would fail in the year 2000. These 18 mission-critical systems were made up of 8,174 programs and data-entry screens as well as some 3,313 databases. With less than seven years to go, someone is going to be working overtime.

By the way, this initial survey required 10 weeks of effort. Ten weeks just to identify the problem areas.

How many systems do you have? How many lines of code do you have in your organization? How many data files? How many maintenance programmers?

The problem extends beyond mere calculations and into the I/O processes of every application. Can you enter 2000 into your data screen, or can you enter only two digits, forcing the input of 00? Can your hard-copy reports print four digits?

The crisis is very real and potentially very costly. Ken Orr, principal at the Ken Orr Institute, and Larry Martin, president of Data Dimensions, Inc., estimate that Fortune 50 organizations will each have to spend about 35 to 40 cents per line of code to convert all their existing systems to accept the change from the year 1999 to 2000.

That translates into about \$50 million to \$100 million for each company. The mind boggles at a maintenance problem with that price tag.

And the costs could be even higher. "The truth is, until we work through a complete cycle with some large organization, we are not going to really know," Orr says.

I have spoken at association meetings and seminars, and when I ask for a show of hands of people addressing the problem, the response is underwhelming. If I get one in 10 respondents, I'm facing an enlightened group.

Typically, all I get are snickers and comments such as, "I won't be in this position or this company in the year 2000. It's not my problem."

This attitude in the computing community is the real problem. It is very difficult for us to acknowledge that we made a "little" error that will cost companies millions of dollars. It is also a "pay me now or pay me later" situation.

"We in the IS industry have not been paying our way," says Gerald Weinberg, author of *Quality Software Management* and winner of the 1991 J. D. Warnier Prize for Excellence in Information Science. "We have been building up a 'national debt' just as surely as the U.S. has been building up a money debt. It will be paid by our

<http://www.year2000.com/archive/NFfw-article.html> 1/10/00

We don't have a choice. We must start addressing the problem today or there won't be enough time to solve it. Status quo means applications that will produce meaningless results in the new millennium.

Weinberg says he believes this procrastination is an indication of deep management malaise. "If software engineering managers cannot manage a change that they've had 1,000 years to prepare for, how can we expect them to manage a change that happens without notice? In other words, if this change causes a crisis in your organization, *everything* will cause a crisis in your organization - and often nothing will cause a crisis."

The inability of the industry to even think about such a project is troublesome. "No one wants to step up to the issue - not [IS] management, not the vendors, not the industry gurus," Orr says. "As with all legacy systems, this problem is messy, expensive and unromantic. No one wants to go in and tell management they have a multimillion-dollar requirement just to keep the business running and that they really have no options."

The reason that nothing is being done, says Capers Jones, chairman at Software Productivity Research, Inc., is that the software industry isn't used to taking long-term preventative steps. "I expect that most companies will not start worrying about the problem until 1999," Jones says. "For some, this will be too late."

Now the good news

There is good news. Object-oriented systems may be able to help. Faced with the huge maintenance costs of fixing their systems, firms may opt to rewrite systems from scratch using object-oriented programming techniques. Tom Love, IBM vice president of the Object-Oriented Group, is a proponent of this theory.

Some companies are unveiling testing and inventory tools that may ease the identification of trouble spots.

Others are hoping that bombarding people with information is the best remedy. To that end, William Goodwin in Brooklyn, N.Y., publishes a newsletter entitled "Tick, Tick, Tick," which brings together people in the IS industry concerned about the impact of the year 2000.

But is the warning falling on deaf ears? "I feel like a lone voice crying in the wilderness," says Brian Pitts, one of Goodwin's subscribers and a project manager at Berry Co. in Dayton, Ohio. "Current economic conditions are making this problem more difficult to address. Management is focused on short-term results and is placing long-term negative consequences on the back burner."

The next seven years will be filled with dire predictions. "You are going to become very, very tired of millennium moaners telling you that your software will fail as it enters the new millennium," says Nicholas Zvegintzov, publisher of *Software Maintenance News*. "But be patient with them. There really is something to be said for them."

Peter de Jager is a speaker on year 2000 computing issues along with the topics of change, creativity, and management technology. Speaking requests should be sent to bookings@year2000.com.

To support Year 2000 awareness and education, Peter has authored a book, two video tapes, and an audio tape. Information on Peter's book, "Managing 00 : Surviving the Year 2000 Computing Crisis" is available from Amazon.com using [this link](#).

Information on Peter's video tapes, "The Year 2000 Challenge: Can we fix it in Time?" and "Year 2000: Are You Ready?" is available through [this link](#), and information on the 60 minute audio tape "Systems for the year 2000: The Upcoming Date Crisis" is available through [this link](#).

Many of Peter's other writings on the Year 2000 computing crisis are available through the archives section of this web site. To access Peter's non-Year 2000 related articles, visit www.newzletter.com.

Vendors	Archive	Jobs 2000	Products	Announce List	Y2K Home
Search	User Groups	Links	Y2K WIRE Press Services	Conferences	Y2K Stocks

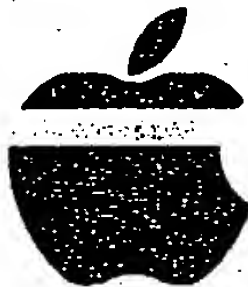
<http://www.year2000.com/>

NEWS

- Lotus resists user pressure for a Notes runtime option but promises improvements in service and application development tools. *Page 4*
- A new crop of distributed network management products aims to overcome SNMP's limitations in enterprisewide management. *Page 6*
- More than 75 leading Unix vendors unite to create a common set of application programming interfaces. *Page 8*
- Digital makes a bid for wider Alpha support with the introduction of a Pentium-class chip for \$3,000 to \$4,000 Microsoft Windows NT PCs. *Page 10*
- Extra! Microsoft begins to act on its promise to tie a messaging function into Windows. *Page 12*
- Unable to meet demand for the Thinkpad, IBM PC Co. announces a new line with a different, and more easily obtainable, screen technology. *Page 14*

DESKTOP COMPUTING

- Apple uses the Macintosh operating system as a lure to build interest in the PowerPC platform. *Page 37*



WORKGROUP COMPUTING

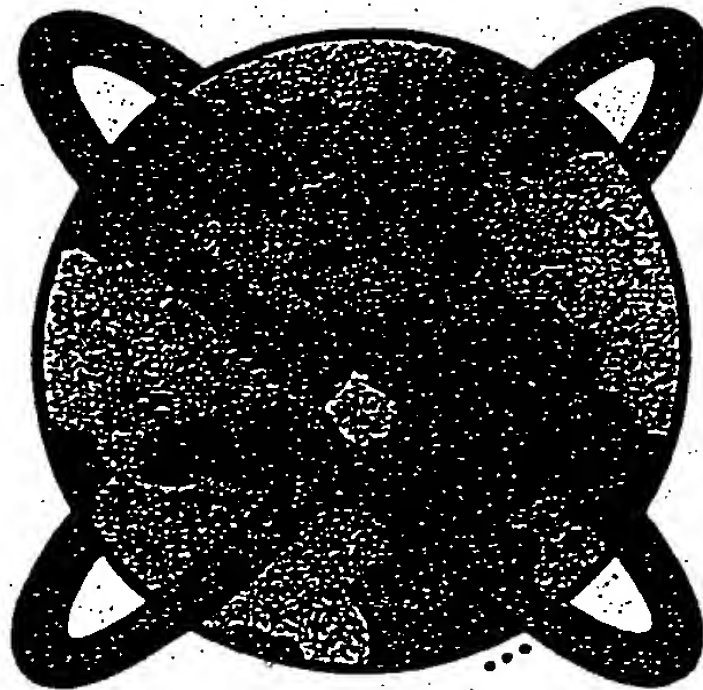
- Oracle finally agrees to support Banyan's Vines network operating system, but not with its latest link product. *Page 51*

ENTERPRISE NETWORKING

- Users and vendors agree that Simple Network Management Protocol needs help with enterprise networks. *Page 63*

LARGE SYSTEMS

- Long-awaited commercial applications for massively parallel processor computing finally arrive. *Page 71*



Enterprise Networking: The Internet is bringing life-saving information to developing nations. Page 63

- IBM is about to introduce an automated tape library for the AS/400. *Page 71*

APPLICATION DEVELOPMENT

- Computer Associates is winning over previously skeptical Clipper users. *Page 81*

MANAGEMENT

- Salaries are up slightly, according to *Computerworld's* seventh annual Salary Survey, although largesse more often takes the form of merit bonuses. *Page 91*

IN DEPTH

- In a little more than 2,000 days we'll flip our calendars to the year 2000. Many mission-critical systems may not survive the date change. *Page 105*

COMPUTER INDUSTRY

- Control Data Systems celebrates its first year of independence and four profitable quarters. *Page 133*

CAREERS

- Flexibility is one of the draws of contract programming, but sign up with the wrong broker and you can kiss that benefit goodbye. Some agencies have been criticized for excessively restrictive contracts. *Pages 113 and 117*

MARKETPLACE

- Household names they aren't, but IS executives tell us these products are great. *Page 126*



COMMENTARY

- RDBMS vendors aren't rushing to provide the support for objects that leading-edge companies need, Charles Babcock observes. *Page 6*
- Editor-in-chief Bill Laberis suggests that the relative improvement in women's pay seen in *Computerworld's* 1993 Salary Survey may mean downsizing is squeezing out discrimination. *Page 32*
- Tim Lynch at Florida Power and Light says OOP has its place, but it isn't the miracle advocates would have you believe. *Page 33*
- Patricia Seybold says you can't have an information-rich organization until you convince people to share. *Page 33*
- Wait for the simplicity that vendors always promise with the next release, Ellis Booker writes, and opportunity will pass you by. *Page 74*

Calendar.....	Page 104
Company Index	Page 131
Editorial/Letters to the editor.....	Page 32
Friday Stock Ticker	Page 132

Executive Briefing

Good news, but... could describe a lot of the news this week, starting with this nugget from our seventh annual Salary Survey: CIO salaries are climbing slightly after an ominous dip last year. Less encouraging is the consensus that downsizing and streamlining will continue unabated, and CIOs who don't keep up technologically are prime targets. *Page 91*

More mixed messages in terms of technology developments: Leading Unix vendors have announced their intention to create consistent APIs for Unix client/server applications, which could mean lower costs for users in terms of packaged software as well as training and integration. Only problem is, the

APIs won't appear until the middle of next year, and it could be two years before customers see real benefits. *Pages 1 and 6*

Token Ring users might be happy to hear that an IEEE working group is taking up the issue of how to speed up the Token Ring architecture. FDDI and ATM are

already options for those who require more speed, but both entail changing hardware and cabling. Some observers are concerned, however, that this effort, in which IBM and Proteon plan to participate, will produce more confusion than actual benefit. *Page 54*

Companies that have ventured into use of SNMP for large-scale network management say they are already plenty confused. No matter what it's called, working with this protocol is not simple. DHL Worldwide had to develop agent technology to handle local polling and install a utility so devices could send periodic "I'm here" signals because messages were being lost enroute to Hewlett-Packard's SNMP-based OpenView. *Page 63*

Users are chafing at the high price of Lotus Notes and opting for lower priced E-mail applications that offer hooks into Notes. *Page 4*. Meanwhile, new competition has appeared on the horizon. Oracle, which once huddled with Lotus on the subject of linking Notes with the Oracle database, has decided to move ahead with a full-fledged product of its own. *Page 1*. And Microsoft is working on integrating E-mail with its "Chicago" version of Windows, which would strengthen its hand in the workgroup area. *Page 12*. Despite customer protests and the specter of competition, Lotus is standing firm on pricing and offering only long-term improvement to its application development environment. It also has a new service option. *Page 4*

In Depth

BY
**PETER
DE JAGER**

Have you ever been in a car accident? Time seems to slow down as you realize you're going to crash into the car ahead of you.

It's too late to avoid it — you're going to crash. All you can do now is watch it happen.

The information systems community is heading toward an event more devastating than a car crash. We are heading toward the year 2000. We are heading toward a failure of our standard date format: MM/DD/YY.

Unfortunately, unlike the car crash, time will not slow down for us. If anything, we're accelerating toward disaster.

THE COST FOR PROGRAMMING TO ADJUST ALL SYSTEMS FOR THE YEAR 2000: \$50 BILLION.

This is a good news/bad news story. First the bad news: There is very little good news. There is no way to avoid the fact that our information systems are based on a faulty standard that will cost the worldwide computer community billions of dollars in programming effort.

Perhaps more importantly, we are going to suffer a credibility crisis. We and our computers were sup-

The date change in the year 2000 — an event that may trigger fatal errors in mission-critical systems — is only 2,308 days away. Many IS people are unprepared or unconcerned.

posed to make life easier; this was our promise. What we have delivered is a catastrophe.

The problem is twofold: the date issue itself and, more importantly, our reluctance to address the problem.

Problem ID

What exactly is the "problem"? To save storage space — and perhaps reduce the amount of keystrokes necessary to enter a year — most IS groups have allocated two digits to the year. For example, "1993" is stored as "93" in our data files, and "2000" will be stored as "00." These two-digit dates exist on millions of data files used as input to millions of applications.

This two-digit date affects data manipulation, primarily subtractions and comparisons. For instance, I was born in 1955. If I ask the computer to calculate how old I am today, it subtracts 55 from 93 and announces that I'm 38.

So far so good. But what happens in the year 2000? The computer will subtract 55 from 00 and will state that I am -55 years

Doomsday, page 108

THE COST WHEN YOU ADD DESIGN, MANAGEMENT, HARDWARE, SOFTWARE AND SUPPORT: \$75 BILLION.

Source: Cost and programmer time estimates come from Bill Goodwin, editor and publisher of "Tick, Tack," a newsletter for people concerned about the impact of the year 2000.

DOOMSDAY

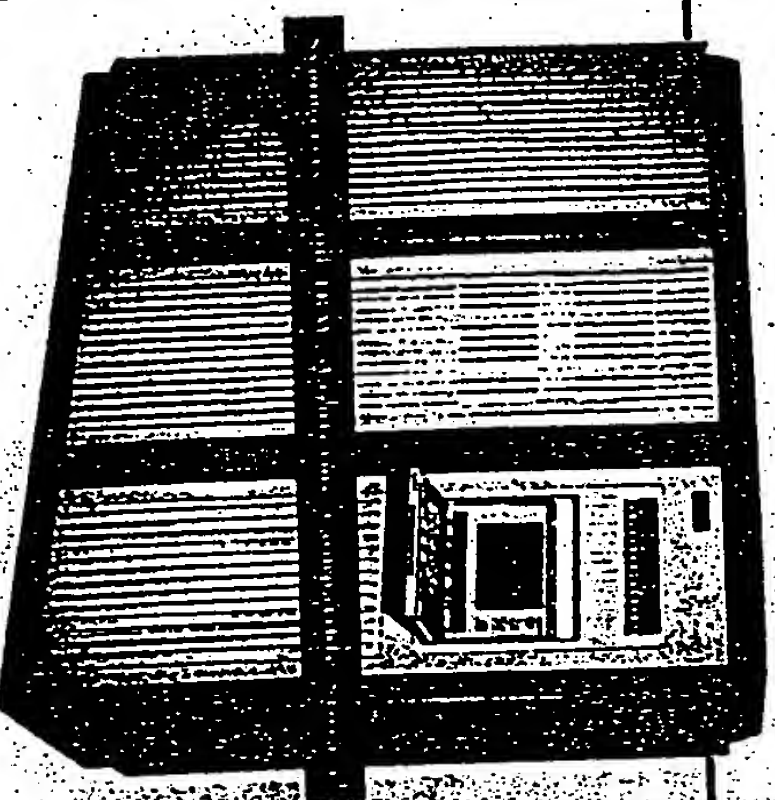


While other mainframe disk companies have been slow to deliver high data availability, the new Symmetrix 5500 ICDA™ from EMC

was designed for IBM and compatible mainframe sites that need continuous operation — 24 hours a day, seven days a week. With redundant hardware

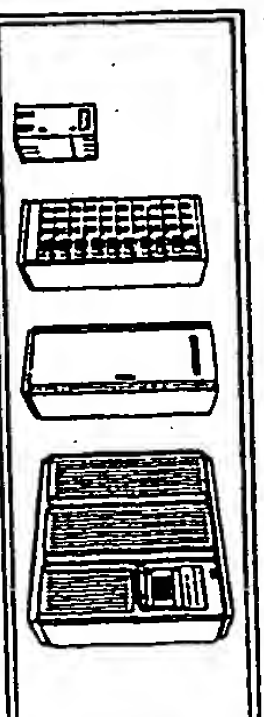
Never before a disk storage fast been

now brings the unbeatable combination of superior performance and continuous operation to high-end mainframe computing. What's more, EMC is shipping this product today. The Symmetrix 5500 is the latest evolution of the high performance Symmetrix Series of Integrated Cached Disk Arrays (ICDA™), and



fore has e system this o available.

components, proactive maintenance features, a full mirroring option and the ability to repair or upgrade the system with no loss of uptime, the Symmetrix 5500 offers the highest level of data availability you can find in the market today. And, the Symmetrix 5500 incorporates the high performance, small



The Symmetrix 5500 is part of an entire line of disk storage systems based on EMC's Integrated Cached Disk Array (ICDA™).

footprint and low cost of ownership that has become the hallmark of the Symmetrix Series. Over 1,000 Symmetrix installations world-

wide are proof of the widespread acceptance of Symmetrix ICDA™ technology.

To inquire about Symmetrix 5500 availability, please call 1-800-424-EMC2, extension LM64C.

EMC²
THE STORAGE ARCHITECTS

EMC, Symmetrix, Integrated Cached Disk Array and ICDA are trademarks of EMC Corporation. IBM is a trademark of International Business Machines Corporation.

TOOLS FOR 2000

CALENDAR ROUTINES

TRANSCENTURY DATA SYSTEMS
(415) 255-7002
A single set of call routines with a 10,000-year date range from 1 A.D. to 9999 A.D. It offers 52 different date formats, 10 different holiday tables and 2,400 definitions of the workweek.

Package can accept dates with-out centuries by defaulting to the century based on the cutoff year of a company's choice.

GENERAL PURPOSE SYSTEMS

ANALYZER/PCSA
COBOL MAINTENANCE
TECHNOLOGIES
P.O. Box 122000
CHICAGO, ILL. 60612-0000

Helps standardize data areas and rationalize processing routines in mainframe Cobol programs. Basically, it downloads a Cobol program onto a PC and analyzes it using GP/34.

GP/34 lists all data names that appear in the program. The data definitions for each data name are listed and will help determine whether the year is stored in a two or four digit field. The system also offers users a list of all procedure code referring to the data names. Users can see whether there are date comparisons that will cause problems for any data names that were not on the initial list.

PC/34
ANALYZER
(415) 974-0600

The product reportedly helps identify all data occurrences and their primary and secondary relationships. It is said to locate affected data sets, databases, files, data name definitions and data storage locations, as well as data in use at the time of code load.

34/000
SOFTWARE FACTORY, INC.
(404) 967-9117

Billed as a "re-engineering" tool for Cobol and C++ systems, 34/000 is a D-Ray feature that gives users the capability to highlight and operations within a program. This is said to enable maintenance modifications to accommodate the century change.

TIC/PC
ISOCOM CORP.
(712) 967-2424

Designed time testing tool for MVS/ESA and XA that claims to support all languages. It lets users test jobs using different dates with-out requiring separate test systems and without affecting other jobs running at the same time.

Doomsday, from page 105

old. This error will affect any calculation that produces or uses time spans, such as an interest calculation.

If you have some data records and want to sort them by date (e.g., 1965, 1905, 1966), the resulting sequence would be 1905, 1965, 1966. However, if you add in a date record such as 2015, the computer, which reads only the last two digits of the date, sees 05, 15, 65, 66 and sorts them incorrectly.

These are just two types of calculations that are going to produce garbage. There are others.

The task facing us is to identify and correct all the date data and check the integrity of all calculations involving date information. We must correct the data residing in all data files or write code to handle the problem.

The starting point

How do we identify the problem data and the associated calculations? We have few, if any, standards for labeling data used in date calculations. The only choice we have is to examine each line of code and make the necessary changes.

One IS person I know of performed an internal survey and came up with the following results: Of 104 systems, 18 would fail in the year 2000. These 18 mission-critical systems were made up of 8,174 programs and data-entry screens as well as some 3,313 databases. With less than seven years to go, someone is going to be working overtime.

By the way, this initial survey required

10 weeks of effort. Ten weeks just to identify the problem areas.

How many systems do you have? How many lines of code do you have in your organization? How many data files? How many maintenance programmers?

**AVERAGE TIME
NEEDED FOR
CODE 2000:
7 DAYS PER
PROGRAM.**

The problem extends beyond mere calculations and into the I/O processes of every application. Can you enter 2000 into your data screen, or can you enter only two digits, forcing the input of 00? Can your hard-copy reports print four digits?

The crisis is very real and potentially very costly. Ken Orr, principal at the Ken Orr Institute, and Larry Martin, president of Data Dimensions, Inc., estimate that Fortune 50 organizations will each have to spend about 35 to 40 cents per line of code to convert all their existing systems to accept the change from the year 1999 to 2000.

That translates into about \$50 million to \$100 million for each company. The mind boggles at a maintenance problem with that price tag.

And the costs could be even higher. "The truth is, until we work through a complete cycle with some large organization, we are not going to really know," Orr says.

I have spoken at association meetings and seminars, and when I ask for a show of hands of people addressing the problem, the response is underwhelming. If I get one in 10 respondents, I'm facing an

enlightened group.

Typically, all I get are snickers and comments such as, "I won't be in this position or this company in the year 2000. It's not my problem."

This attitude in the computing community is the real problem. It is very difficult for us to acknowledge that we made a "little" error that will cost companies millions of dollars. It is also a "pay me now or pay me later" situation.

"We in the IS industry have not been paying our way," says Gerald Weinberg, author of *Quality Software Management* and winner of the 1991 J. D. Warnier Prize for Excellence in Information Science. "We have been building up a 'national debt' just as surely as the U.S. has been building up a money debt. It will be paid by our children — our successors — one way or another," Weinberg says.

We don't have a choice. We must start addressing the problem today or there won't be enough time to solve it. Status quo means applications that will produce meaningless results in the new millennium.

Weinberg says he believes this procrastination is an indication of deep management malaise. "If software engineering managers cannot manage a change that they've had 1,000 years to prepare for, how can we expect them to manage a change that happens without notice? In other words, if this change causes a crisis in your organization, *everything* will cause a crisis in your organization — and often *nothing*

**TOTAL PROGRAM-
MING TIME FOR
ALL SYSTEMS:
1.2 MILLION
MAN-YEARS.**

Bearings firm goes on offense

**BY
LORY
ZOTTOLA
DIX**

Torrington Co. doesn't have time for seers predicting dire consequences when the final tick of the clock strikes a change to the year 2000. That's because this bearings manu-

facturing company, a division of Ingersoll-Rand Co., is too busy doing something about it.

In 1991, at the suggestion of an employee and as part of its drive toward total quality management, Torrington convened a seven-member team, headed by programmer/analyst Bob Hartman-Berrier, to tackle the century date change issue. The Torrington, Conn.-based company knew that because certain of its programs stored dates by their final two digits, a changeover to the year 2000 could mean fatal errors that might throw its global systems into an uproar.

The group brainstormed about problems areas — the company has a mix of mainframes, minicomputers, PCs and local-area networks — and interviewed businesspeople throughout the firm.

Members took an inventory of in-house and outside software products. They sent a questionnaire to Torrington's 90 vendors (30 mainframe and minicomputer, 60 PC) to gauge product support for

JUST DO IT

Here's what's on Torrington's to-do list to prepare for 2000:

1 ■ Set standards. All in-house and vendor system files and any date fields will have a four-digit year. (Status: Done.)

2 ■ Accommodate legacy systems. Despite its move to a distributed environment, the company will maintain its mainframes. It is actively seeking a mainframe compiler that supports four-digit years, something its current compiler doesn't do. (Status: In progress.)

3 ■ Investigate and buy a data analysis tool to identify location of date fields within programs. (Status: Will

be complete in 12 to 18 months.)

4 ■ Develop bridge modules. Some programs, especially those dealing with forecasting, will need to work with the century date as early as 1996. These "critical" systems (such as payroll and job scheduling) will need bridge modules to handle century rollover. These software fixes are a less time-intensive alternative to changing native code. (Status: Will be complete in 12 to 18 months.)

5 ■ Identify critical systems needing bridge modules and prioritize them. (Status: In progress.)

6 ■ Do it. Put the recommendations in action.

four-digit years.

Vendor responses were encouraging. Of the 90% mainframe and 60% PC vendors that returned the survey, nearly all of them either could accommodate the century date or had fixes in the works.

The in-house inventory showed that 30% of Torrington's programs and 25% of its files and databases required changes. Hartman-Berrier says. At 25 hours per program and 40 hours per database file to reformat, unload and reload, the team

In Depth: Doomsday

'will cause a crisis.'

The inability of the industry to even think about such a project is troublesome: "No one wants to step up to the issue — not [IS] management, not the vendors, not the industry gurus," Orr says. "As with all legacy systems, this problem is messy, expensive and unromantic. No one wants to go in and tell management they have a multimillion-dollar requirement just to keep the business running and that they really have no options."

The reason nothing is being done, says Capers Jones, chairman at Software Productivity Research, Inc., is that the software industry isn't used to taking long-term preventative steps. "I expect that most companies will not start worrying about the problem until 1999," Jones says. "For some, this will be too late."

Now the good news

There is good news. Object-oriented systems may be able to help. Faced with the huge maintenance costs of fixing their systems, firms may opt to rewrite systems from scratch using object-oriented programming techniques. Tom Love, IBM vice president of the Object-Oriented Group, is a proponent of this theory.

Some companies are unveiling testing and inventory tools that may ease the identification of trouble spots.

Others are hoping that bombarding people with information is the best remedy. To that end, William Goodwin in Brooklyn, N.Y., publishes a newsletter entitled "Tick, Tick, Tick," which brings together people in the IS industry concerned about the impact of the year 2000.

But is the warning falling on deaf ears?

expects the work load to be heavy.

Costs have been a little trickier to pinpoint. While Hartman-Berrier says worst-case costs to revamp systems could reach \$3.5 million, the best-case could be one-tenth that figure.

Hartman-Berrier and Ken Even, manager of corporate support, hesitate to commit to a number on project cost because of the firm's move to a distributed setup. "Distributed systems may mean we remove some applications, keep others. If we remove a system, we don't have to change it, reducing our labor efforts



The Torrington 2000 team: (front): Traci Winegar, Alison Anstett, Evelyn Pulazini; (back): John Draghi, Bill Beyer, Bob Bouchard, Hartman-Berrier

and resources," Hartman-Berrier says. "We can take a felt-tip marker and strike programs and files from our list."

His advice to other IS professionals is to "plan, plan, plan and act, act, act." The year 2000 is coming. □

TICK, TICK, TICK

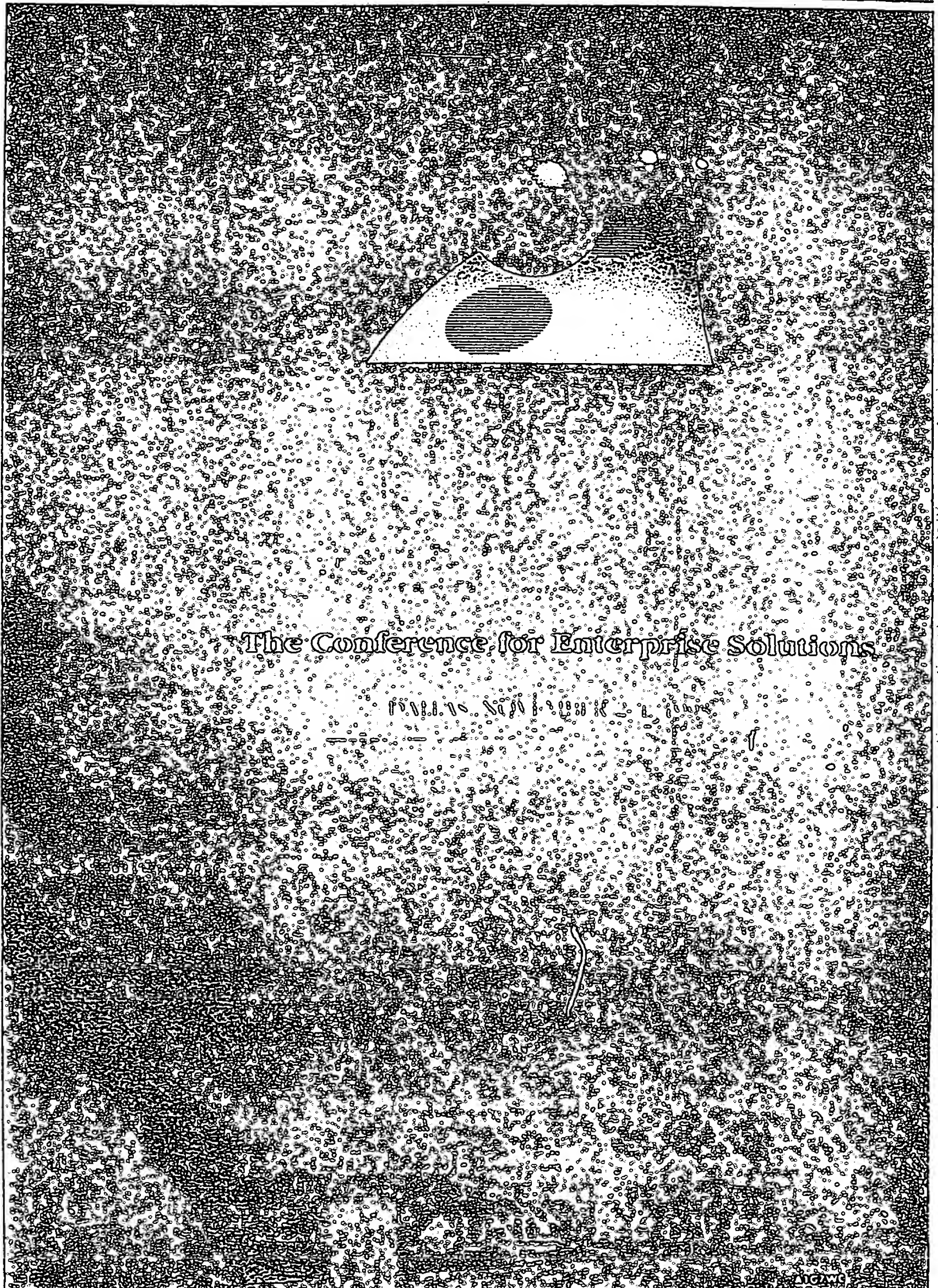
The newsletter "Tic, Tac, Tick," which focuses on the impact the year 2000 will have on systems, is having a Worst-Case Scenario contest. The newsletter is seeking the most creative ideas of what could happen on 01/01/2000. The prize is a free subscription to the newsletter. Send entries to "Tic, Tac, Tick," P.O. Box 020538, Brooklyn, N.Y. 11202-0012 (718) 643-TICK.

"I feel like a lone voice crying in the wilderness," says Brian Pitts, one of Goodwin's subscribers and project manager at Berry Co. in Dayton, Ohio. "Current economic conditions are making this problem more difficult to address. Management is focused on short-term results and is placing long-term negative consequences on the back burner."

The next seven years will be filled with dire predictions. "You are going to become very, very tired of millennium

moaners telling you that your software will fail as it enters the new millennium," says Nicholas Zvegintsov, publisher of *Software Maintenance News*. "But be patient with them. There really is something to be said for them."

DeJager is an industry speaker on the topics of change, creativity and management of technology. He can be reached at (416) 792-5706 or via CompuServe (706112576) and MCI Mail (PDEJAGER).



*13

32

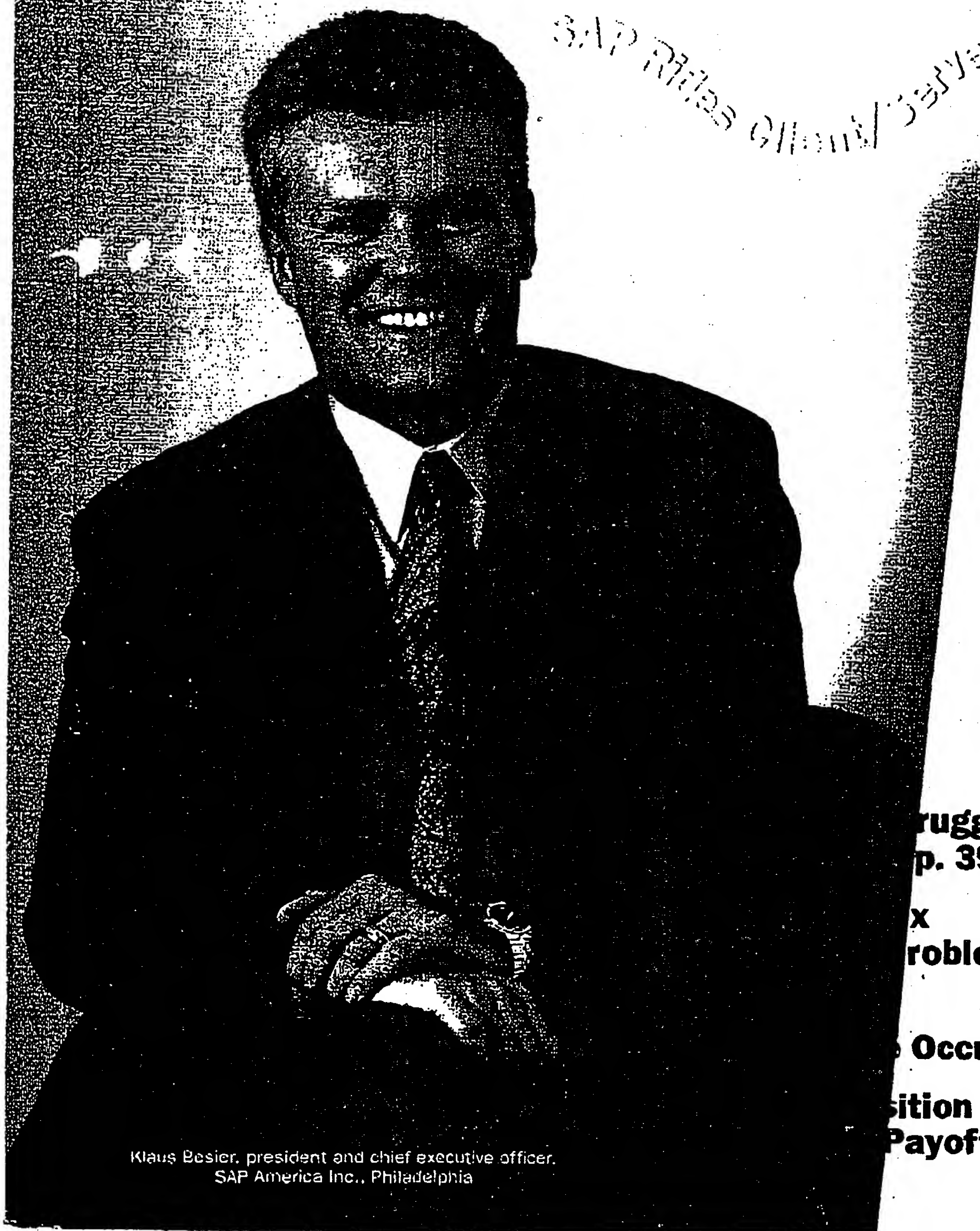
SOFTWARETM magazine

FOR MANAGERS OF CORPORATE SOFTWARE

SEMI-MONTHLY PUBLICATION

APRIL 1995

SAP Rises Client/Server Wars page 104



Klaus Bosier, president and chief executive officer,
SAP America Inc., Philadelphia

Struggle
p. 39

X
Problems p. 59

Occupancy p. 70

sition
Payoff p. 81

FEATURES

COVER STORY

SAP HOLDS EARLY LEAD IN APPLICATIONS MARKET

Michael Bucken

Success has the firm scrambling to keep up with implementation demands

104

SOFTWARE DEVELOPMENT

REPOSITORIES STRUGGLE TO STAND ALONE

Mary Hanna

Tool integration issues hinder their efforts to be enterprise warehouses

39

40 Product Listing: Repository-based Development Tools

ENTERPRISE SYSTEMS AND NETWORKING

PINPOINTING UNIX PERFORMANCE PROBLEMS

Paul Korzenkowski

Unix sites find tools for both historical and realtime data few and far between

59

64 Product Listing: Performance Monitoring and Management Products

DATA MANAGEMENT/DBMS

DATABASES VIE FOR WAREHOUSE OCCUPANCY

Barbara Francett

Relational and multidimensional DBs may end up as OLAP roommates

70

77 Product Listing: Multidimensional Databases & OLAP Servers

CLIENT/SERVER APPLICATIONS

FINANCIALS POSITION FOR WORKFLOW PAYOFF

Colleen Frye

Workflow is a goal, but not yet a priority, for financials departments

81

OPEN SYSTEMS

FLEET NETWORKS PREPARE FOR TAKEOFF

Jerry Cashin

The late '90s will be the proving ground for ATM, Frame Relay and other nets

90

INTERNATIONAL

UNIX IS AUSSIE PASSPORT TO NETWORK UNIFICATION

Keth Power

NSW RTA integrates registries, develops system, using IEF on Unix platform

101

Authorization to photocopy for internal use is granted by Sentry Publishing Co. Inc., provided the appropriate fee is paid to Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, Tel. (508)750-8400. For information or quotations on reprinting articles — please contact Barbara Gagne, Sentry Publishing Co. Inc., One Research Drive, Ste. 400B, Westborough, MA 01581. Tel. (508)366-2031.



Photo from cover story, p. 104

DEPARTMENTS

Editor's Letter	4
Forum	6
Newsfront	14-18

Field Report	
Is OO Cobol Too Late?	25
All's Quiet on Sysplex Front	28
Sun Regroups with Solstice	31

Companies on the Move	
Atria Software	33
Next Computer Inc.	33

News Briefs	34
-------------	----

Industry Briefs	36
-----------------	----

New Products	110-115
--------------	---------

Marketplace	115
-------------	-----

Calendar	116
----------	-----

Advertisers' Index	116
--------------------	-----

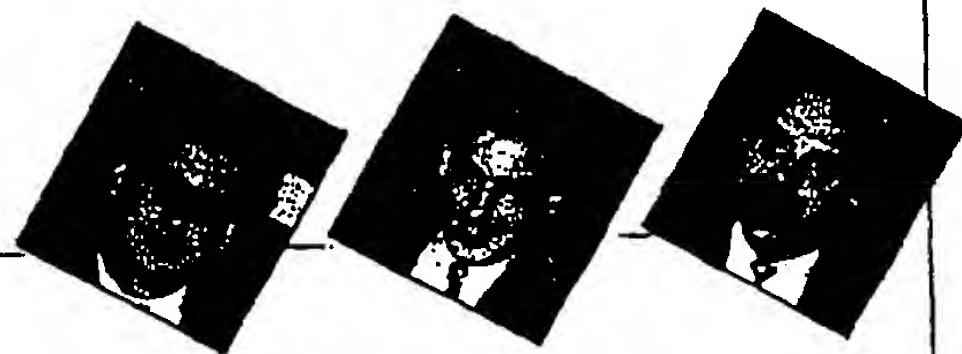
Our Back Pages	118
----------------	-----

COVER PICTURE

Cover photo and photo on p. 105 by Ed Lederman, New York City.

Party When It's 1999

Jeff Furman, Albert Marotta & Cliff Candiotti



DATELINE: December 31, 1999

You are piloting an F-22 above the Pacific Rim. It is one second to midnight and the foreign craft tracking you is so close you're obliged to send a warning signal as a New Year's greeting.

The other pilot has two seconds to respond. Your onboard strategic systems are now calculating the time difference between when you sent your message and when a reply will reach you.

You wait. The interval seems interminable. To your equipment, though, it is extremely short: -1.5 seconds to be exact! Because you sent your signal in the year "99," and received the reply in the year "00," the difference is negative, and your weapons system is arming!

As the year 2000 looms, MIS must come to grips with the task of identifying century-change bugs across all systems, fixing applicable application components, and testing these applications.

Software vendors are developing products to help users face the millennium. These fall into five categories:

1. System date simulators. These tools allow IS to pass user-specified system dates to batch jobs, TSO user IDs and CICS components. If systems that depend intensively on date-based processing or that routinely run across date boundaries or time zones have already been identified, a date simulator might be the most powerful weapon in a year-2000 arsenal.

Users can set up duplicate tests simulating date changes in key processing windows, giving "before" and "after" results.

Date simulators generally permit altered dates and/or times to be passed to jobs based on criteria like job name, job name pattern, job class and job submitter ID. They can be customized to allow simulated dates and times to be passed to TSO sessions and started tasks. All date simulators have security measures providing central control over which jobs are tested under the simulator.

Systems programming groups should assess these products for their impact on overall system performance, including all jobs that do

If organizations prepare for the year 2000, they'll be able to enjoy their century-end bash, instead of making a midnight run to a disaster recovery site.

heavy system date queries. These products work by intercepting SVC 11 calls, and even jobs that do not request changed dates or times will be affected if they run on the same CPU as an active date simulator.

2. Code analyzers. Though date simulators provide instant gratification by quickly exposing date-based bugs, source analyzers can be a real

workhorse in assessing year 2000 retrofit requirements. This type of product is a logical jumping-off point for those who need to get a handle on the system-wide scope of their conversion effort.

Code analyzers are able to scan source management configurations for date-sensitive programs, the JCL and procs that run them, the files they impact, and the business processes affected. Some also provide information on dependencies, interfaces, architectures, strategies, business rules and formulas. Product design ranges from mainframe-based data dictionaries and repositories, to data models and Case tools developed on client/server platforms.

There are analyzers specifically tailored for year 2000 conversion efforts, as well as generic impact analysis tools adaptable for this purpose. One product, for instance, allows users to create data tables of date-related Cobol keywords, run scans against source code based on the tables, and produce a cross-reference report for each program. Some products allow user-specified searches and updates on a variety of file formats. And some can even analyze regions and address spaces for all system date activity, identifying the best candidate programs for date-simulation testing.

3. Add-on date functions. Another option is pre-written, pre-tested vendor calendar date routines. There are a number of products available that will accurately process dates and leap year situations.

With a date routine product, there is no need to convert existing

Jeff Furman is systems support manager, and Albert Marotta and Cliff Candiotti are systems programmers, at Prudential Securities, New York City.

Forum

databases to four-digit year fields. Typically these products come with hundreds of date-handling utilities callable from various languages and portable across multiple platforms. Most vendor calendar routines expect a two-digit year as input, and can determine which century field to append to the date. The user defines an OVER/UNDER date, and the routines do the rest.

A date routine product can compute the difference between two dates, add/subtract the number of days to/from a certain date, and find "milestone" dates, such as the "end of the next fiscal quarter."

The decision to add this functionality generally hinges on impact analysis identifying which programs will fail. Adding calendar function calls to application programs is a way of thwarting problems with a minimum of code changes. Adding a call to any such calendar function, however, requires coding, compiling, linking and testing, ideally with a date simulator. This may mean dredging up source code that has not been touched in years.

4. Language upgrades. In an attempt to make legacy systems century-sensitive, many IS organizations have used manual methods — moving values to a century field, for example. In addition, many applications still contain homegrown routines for retrieving dates, converting date formats and performing date-based calculations. Such nonstandard methods may contain time bombs.

The best thing to do is to upgrade to a language that comes with year 2000 processing capabilities, such as Cobol/370, C, Rexx or client/server development tools. Cobol/370 comes with a century-sensitive four-digit year function, and boasts a number of other new routines. These perform many date-related conversions and calculations, as well as numerous math functions, while retaining the consistency of Cobol syntax.

There are, of course, downsides to upgrading. Resistance to change is one. Also, Cobol/370 has some compatibility issues that can make a

complete upgrade prohibitive.

One alternative is to phase in new software so it exploits Cobol/370 functionality. Programs are compiled with the Cobol/370 compiler; executions reference the Cobol/370 runtime libraries in a STEPLIB DD statement. A second solution is to write Cobol/370 subprograms that are called by legacy Cobol systems.

5. Database converters. At some point in the process of readying applications for the 21st century, users will need to store the first 2 bytes of the year. This means that a date field in a database will need to grow to 8 bytes from the traditional 6 (Display Mode). Since users don't want to re-create every database in their shop, they need a way to convert existing files.

Some products designed to manipulate file data can be used for storing century fields. This type of utility usually works with a "before" and "after" set of copybooks, where the "before" is the one used in programs that reference the file, and the "after" is what the record should look like after the conversion. The good news is that a product like this may already be in-house. The bad news is that this method brings up a number of issues: How do I initially insert the century digits into the record? Won't this increase the size of each record? Will every field after the date be in the wrong position relative to the start of the record? What if the file is Vsam, and the date is part of the key? If I have several record types within one file, how do I convert only the date records?

There is at least one product available that deals with such issues, and was designed specifically for century change solutions. An "intelligent" database converter lets users decide how data should be converted. For example, if users do not want to change the length of a record, they can add 2 bytes to a date field by taking 2 bytes from a "Filler." Or, they can tell the utility to change the format of the date field; for instance, from "Display," to "Packed" or "Binary." This increases the number of

digits but not the number of bytes.

No Midnight Run

It is always better to deal with problems before they arise. Nowhere is this more true than in data processing, and the year 2000 holds the possibility of some major threats. Complications can manifest in a number of ways. Programs might not know it's a leap year, wind up with negative numbers instead of positive, move the wrong century indicator, treat the year as lower than the previous year instead of higher, or just fail mysteriously.

It is no exaggeration to say that every date-processing program in the world — from mainframes to minis to the latest PC systems — is vulnerable to error. And errors might be more than just invalid reports. A wayward program might lose or misdirect financial figures, for example. And software that controls tangible objects — like ATMs, airplanes or even an occasional smart bomb — could bring more than fiscal ruin.

After evaluating the various types of products designed to help companies face the millennium, we feel no one product is a complete solution. They complement each other. We expect to see more sophisticated, and possibly more comprehensive, products released as part of this new "cottage industry," and we believe they will play a key role in maintaining year 2000 integrity in many shops. Other help is also on the way, in the form of year 2000 newsletters, conferences and training sessions.

If organizations prepare for the year 2000, they'll be able to enjoy their century-end bash, instead of making a midnight run to a disaster recovery site. ■

CORRECTION

An editing error in the February Editor's Letter gave the impression that the Essbase multidimensional database was jointly developed by the founding members of the OLAP Council. Essbase was developed by and is sold only by Arbor Software.

COMPUTERWORLD

\$2.00 A COPY; \$44/YEAR

FEBRUARY 13, 1984

VOL. XVIII, NO. 7

In Depth

C.J. Date
defends relational
system perfor-
mance ID/15

Lost in space

Satellite launch
goes awry/11



Thomas Nies
Cincom's chief
talks about data
base trends/17

SECTIONS

Editorial/60

Software &
Services/69

Communications/91

Systems &
Peripherals/99

Microcomputers/107

Computer
Industry/131

Investors sue after STC ices CPU project

SAN FRANCISCO — Storage Technology Corp.'s recent cancellation of a project to develop an IBM-compatible processor resulted in the filing in federal court here last week of a class action suit charging the company with securities fraud and racketeering.

The suit, filed by the Chicago law firm of Katten, Muchin, Zavis, Pearl & Galler, charges STC with stock fraud and seeks actual damages of \$10.6 million and punitive damages of \$20 million. It also seeks unspecified additional damages that may be tripled under the provisions of the federal Racketeer Influenced and Corrupt Organizations Act, a statute generally used by the federal government to prosecute organized crime figures.

Donald E. Egan, the attorney who filed the suit, charged that STC had created a partnership to finance the processor project and, four months before canceling the project, obtained \$10.6 million from 181 new limited partners who were given "written statements and misrepresentations of material facts that were untrue or misleading."

Also named in the suit was the investment firm of Smith, Barney, Harris, Upham and Co.

STC killed the project to develop a high-performance, IBM-compatible mainframe after citing technological problems and increasing cost estimates [CW, Feb. 6].

At press time, STC had no comment.

Supermini hits campaign trail with Cranston



CAMPAIGN '84

By James Connolly
CW Staff

WASHINGTON, D.C. — What may be the most sophisticated in-house computer system ever used by a presidential candidate is helping U.S. Sen. Alan Cranston (D-Calif.) in his try for the Democratic nomination.

Cranston, ranked as low as sixth in polls that show him more than 40 percentage points behind former Vice-President Walter Mondale, apparently is the only one of eight candidates to use a minicomputer and the first to use a supermini in his campaign headquarters.

Like the competing organizations, the Cranston campaign uses computers for mass mailings of political and fund-raising letters, for scheduling and for income and expense reports for the Federal Election Commission. But while the other campaigns use service bureaus and in-house microcomputers, the Cranston for President Committee has a Digital Equipment Corp. VAX-11/750 in its headquarters here and DEC Rainbow micros in its New Hampshire and Iowa field offices.

"We made a decision well back in 1982 to go in-house," Alex Thurber, the committee's director of information systems, said. "A campaign doesn't have the time to wait for someone else to do its processing. We

See CRANSTON page 4

M&D offering tool to build applications

By Paul Gilm
CW Staff

NATICK, Mass. — McCormack & Dodge Corp. last week continued its foray into the systems software market by releasing as a package the system development software used to build its Millennium application series.

Millennium:SDT can be used to build IBM CICS applications that are fully integrated with mainframe-based M&D Millennium applications, including general ledger, fixed assets, human resources, accounts payable, purchase order and capital project analysis.

The package incorporates both interpretive and generative capabilities as well as reusable code, M&D said. Application definition is accomplished by creating data base records for files, records and fields, screens, queries and search fields.

M:SDT programs can call literals, variables and parameters and map them directly to the screen, eliminating CICS mapping and security, according to an M&D spokesman.

M:SDT uses M&D's procedure definition language, Millennium:PDL, to access and process data within the data bases. M:PDL uses an English-like syntax and arithmetic and logical operators to execute complex expressions.

On-line features offered with M:SDT and all Millennium products include multi-
See M&D page 6

PROFILE

He heads 'two-in-one' department

By James Connolly
CW Staff

SMYRNA, Tenn. — Logic says it is a mismatch, a marriage in which the partners have little in common and little to say to each other.

But Nissan Motor Manufacturing Corp. U.S.A. says that upon closer inspection, the "mismatch" makes sense.

Nissan merged its information systems with its purchasing department.

The apparent illogic lies in the fact that the departments cover so little common ground. But "we put the two groups together because they don't go together," emphasized Robert A. Frinier, who became vice-president of purchasing and information systems on Jan. 1.

"We think we figured out a way to have systems report to a division where it can be run without bias," Frinier explained. "Purchasing in this facility doesn't place much of a demand on the [DP] systems, at least much less than finance or production control. Our consultants said the systems organization needs a place where it can get top-level support and not be biased by the person in charge being primarily responsible to other areas."

That bias, he noted, could result in a department head tailoring information systems to his own needs at the expense of other departments or paying too little attention to information systems.



Frinier

"We think this concept might knock the computer industry on its ear," according to 42-year-old Frinier, who has a background in production control. Formerly with Ford Motor Co. in Michigan, he has been with Nissan since 1980, working with system users in production and with consultants and technical personnel in developing flowcharts for system interfaces between the new facility here and Nissan's headquarters in Japan.

Frinier heads a department of 70 people, some of whom previously worked in information systems when it was controlled by the Finance Division. Responsibility for information systems was shifted from the Finance Division on Jan. 1, at a time when company President Marvin

See NISSAN page 2

TOP OF THE NEWS

Victor Technologies, Inc. has earned the unhappy distinction of becoming the third microcomputer maker in six months to file for protection under Chapter 11 of the Federal Bankruptcy Code. Page 2.

The Federal Communications Commission modified access surcharges for Foreign Exchange and Wats users, but retained the \$2/mo and \$6/mo fees per line for Centrex-CO. Page 2.

Digital Equipment Corp. has become the first major computer vendor to enter the off-site storage and recovery business. Page 4.

DP salaries are up sharply from six months ago, a recruiting firm reported. Page 9.

Supercomputer maker Denelcor Corp. has jumped on the Unix bandwagon. Page 69.

The Goodyear Blimp is not the only big thing for which the company could become famous. Goodyear Aerospace Corp. researchers claim they have developed a computer system with the fastest I/O rate in the world. Page 99.

C
4810601VHUIVYMA FCBME
UNIVERSITY MICROFILMS INTL
SERIAL PUBLICATIONS
300 N ZEEB RD
ANN ARBOR MI 48106

NEWS

The problem you may not know you have

There is a bug in every Cobol program in your library.

You probably don't know about it, and you almost certainly haven't had to worry about it yet. But when it shows up, it will hit your entire shop, reducing data entry procedures to a mush of error messages.

By Paul Gilpin
CW Staff

NOVI, Mich. — When systems analyst William Schoen has presented DP managers with that pitch, he has understandably piqued their interest. But he has had trouble getting them to take his case seriously when they find out what the bug is.

The problem is the year 2000. The turn of the calendar presents a procedural issue that is acknowledged occasionally in trade conference jokes and DP shop banter but has attracted little serious attention in the industry.

The root of the issue is the industry-wide standard of using two-digit date fields in Cobol programs. Error-checking procedures typically rely upon dates proceeding sequentially, with one year's date being greater than that of previous years.

However, programs will be thrown

for a loop after the turn of the century when they have to cope with the two-digit date field "00" being greater than "99." Schoen ticked off the problems that will arise if the issue is not dealt with: "Program logic is going to malfunction all over the place. Sequential data processes are going to abend. A great many on-line modules won't accept the new dates because they include sequence checks. Most sorts won't work, and a great many literals won't work."

Perhaps understandably, the issue

has received little serious attention from a DP community that is concerned with more immediate problems. But Schoen claims data processing should start paying attention now to the problem of making programs "year 2000-compatible" in order to avoid headaches when the time of reckoning draws near.

He rationalizes that most large firms maintain a library of thousands of Cobol programs, most of which have at least one date field. In addition, many of those programs

have literals, sequence processes and error checks interspersed throughout, meaning that they will require modification in several places.

"You can't assume every date field has the word 'date' in it," Schoen said. "You also can't assume every field that looks like a date field is a date field. You're going to have to look at every program and figure it out."

For that reason, he said, the "black box" approach of simply running every program through a modification routine is impractical. Some date fields are bound to be missed.

See 2000 page 8

200???

S M T W T F S

Manager gets 16-year jump on Cobol bug

TOLEDO, Ohio — Although he agrees that making Cobol programs "year 2000-compatible" is not an issue of immediate concern, a DP manager here has elected to become the first user of a set of subroutines designed to make the transition to the new millennium a smooth one.

Michael Ehrick, director of computer services at Libbey-Owens-Ford, Inc., became interested in preparing Cobol programs for the inevitable march of time after he received a direct mail advertisement from William Schoen, a systems analyst based in Novi, Mich.

Schoen's Charmar Enterprises, Inc. markets a product that addresses the problems that may emerge when the year 2000 arrives and plays havoc with date fields in existing Cobol programs (see related story). His product is "a subroutine that does some elegant arithmetic routines that let you compare date fields with two-digit years in such a way that the year '00' is greater than the year '99,'" Ehrick said. Schoen also offers an approach to "sorting so that the same end is achieved, according to Ehrick.

Ehrick put Schoen in contact with Libbey-Owens-Ford's manager of application development. The company expects to begin including the subroutine in new applications as soon as the next development project is undertaken.

The subroutines are "nothing an advanced computer science major couldn't do," Ehrick said. But, he added, "a lot of creative thought went into this, along with a lot of dogmatism."



Are your backups running into your prime shift?

Incremental Backups can reduce your backup time by 50 to 80%.

User can specify that ABR is to backup changed data sets only until a limit count is reached. At that time, ABR will interpose a full volume dump. This technique greatly reduces backup time while providing a periodic image of the entire volume.

- Automatic Volume Reconstruction
- Preallocation
- Automatic Tracking of Backups
- Unlike Device Support

FDR™

AUTOMATIC BACKUP & RECOVERY

Available for IBM OS/VS & MVS
90 Day Free Trial



INNOVATION
DATA PROCESSING

970 Clifton Ave. Clifton, NJ 07013 • 201-777-1242

NEWS

Victor petitions for Chapter 11

Third micro maker in six months to file

By Patricia Keefe
CW Staff

SAN JOSE, Calif. — Bowing to pressure from its many creditors and several lawsuits, Victor Technologies, Inc. last week voluntarily filed for protection under Chapter 11 of the U.S. Bankruptcy Code.

With an estimated \$70 million in unsecured debt, Victor became the third microcomputer maker in six months to seek court protection.

Victor said its manufacturing unit, which is also named Victor Technologies, Inc., has consented to a separate, involuntary bankruptcy petition filed last week in federal bankruptcy court here by six creditors, whose unsecured claims total over \$12.8 million. The latter petition sought to force the California-based manufacturing unit into Chapter 11, where it could reorganize and work out a debt schedule under court protection from creditors' lawsuits.

According to a statement released by Victor, two other units also filed voluntary petitions: Victor United, Inc., the U.S. distribution subsidiary,

and Victor Technologies International Sales Corp.

The vendor's European and Canadian distribution operations are insulated against claims against the U.S. entities, Victor said.

Victor has reached an agreement with its largest secured creditor, Security Pacific National Bank, under which the bank will finance "near-term manufacturing and distribution needs," a spokeswoman for Victor said. That agreement is subject to approval by the bankruptcy court.

Victor said it will continue to supply an adequate amount of computers to its U.S. and foreign distribution outlets.

As part of a restructuring of operations, Victor has terminated 300 employees in the U.S., leaving approximately 570 employees worldwide, the spokeswoman said. Victor declined to comment further.

Second suit

Another suit filed last week, this one by the advertising and public relations firm of Doyle Dane Bernbach

International in federal bankruptcy court in Manhattan, seeks \$8.3 million — \$3.3 million for services performed (media buying) and \$5 million in damages. That suit accuses Victor of providing false financial statements that overstated the micro maker's true worth and failed to disclose losses suffered in 1982, according to spokesman Robert Frost. The suit also names as a defendant Kidde, Inc., which owns a majority share of Victor, and Kidde President Fred R. Sullivan.

Victor has had a tough two years financially, suffering losses of approximately \$3 million in 1982 and \$47 million to date in fiscal 1983. Its overall debt reportedly tops \$100 million.

The Chapter 11 filing highlights the failure of a creditors' committee established in late 1983 to help Victor devise a plan for paying a debt then estimated at \$90 million. Negotiations between Kidde and the committee reportedly broke down, resulting in a successful bid to force Victor into Chapter 11.

He noted, "Initially, I'm spending 80% to 90% of my time on information systems, learning as fast as I can."

"I have people working for me who have been with information systems for a while. So they're teaching a newcomer," he said.

Access fee cut for Wats, kept for Centrex

By Phil Hirsch
CW Washington Bureau

WASHINGTON, D.C. — The surcharges for accessing the dial-up telephone network will remain the same for Centrex-CO (Central Office) users, but the surcharges for Wats and Foreign Exchange services will be modified, the Federal Communications Commission ruled earlier this month.

Immediately afterward, Jack Smith, chief of the FCC's Common Carrier Bureau, indicated he is still hoping for some reduction in AT&T's upcoming long-distance rates. He said he "would like to see something in the neighborhood of" a 5% to 8% reduction in the proposed rates for Message Toll Service (MTS).

The FCC's latest action on its access surcharge plan was taken at a Feb. 3 meeting. Specifically, the commissioners:

- Reaffirmed an earlier decision under which Centrex-CO users whose systems were in place on or before July 27, 1983 will pay an access surcharge of \$2/mo per line, while those using newer systems will pay \$6/mo per line. The fees will begin April 3 and continue until sometime in 1985 when, depending on the outcome of an upcoming FCC study, they may be altered.

- Converted the closed end of each Wats line from an interstate to an intrastate facility. Under the earlier scheme, a user would have paid a direct charge of \$26/mo per line termination beginning April 3. Now the user will pay the local telephone company a maximum \$6/mo per line access surcharge, plus a varying amount to AT&T based on Wats usage. The FCC anticipates, however, that the latter fee will be absorbed rather than passed on by the phone company, a spokesman said.

- Restructured Foreign Exchange access surcharges so that at least some users of this service will pay less.

The commission decided that a telephone operating company providing local exchange facilities at the "open" or remote end of a Foreign Exchange circuit can charge only 45% of the MTS/Wats access fee rather than an equivalent amount. Although this change will tend to drop Foreign Exchange rates below what they would have been, the total bill is likely to go up significantly when compared with present rates because of pending increases for jurisdictionally interstate short-haul Foreign Exchange access circuits.

However, rates for similar long-haul circuits of 25 miles or more generally have not been increased and in some cases have been lowered, the spokesman said.

One problem that remains to be resolved, the spokesman added, involves local telephone companies not equipped to meter usage. In the past, they have charged each Foreign Exchange user for 4,400 min/mo of use, which forced those with lower calling volumes to subsidize those with higher volumes. The commission is studying this problem and a decision is imminent, the spokesman said.

NISSAN from page 1

Runyon installed new vice-presidents for product control, engineering and planning and finance as well as for purchasing and information systems.

The highly automated plant has

been producing an average of 7,000 light trucks a month since beginning production last June, according to Frinier.

"I don't have a very strong computer background from a technical point of view," the Nissan executive acknowledged.

NEWS SUMMARY

Digital Equipment Corp. has entered the off-site storage and disaster recovery business 4

Honeywell, Inc. reduced the purchase prices of some DPS 8 and DPS 88 mainframes 4

An analyst's tool is addressing the programming problems of the year 2000 7

The Postal Service has issued final regulations on submitting mail with nine-digit zip codes 8

Source EDP, a recruiting firm, reported that salaries in the computer profession are up sharply from six months ago 9

The on-board computer failures of the Columbia space shuttle flight have been diagnosed and an additional quality control procedure was established by IBM to prevent a recurrence 10

Two communications satellites intended to transmit voice, data, video and facsimile traffic failed to reach proper orbit after their launch from the space shuttle Challenger 11

The U.S. Air Force debuted the first of 153 Sperry Corp. mainframes being installed under the largest computer contract in history 12

Digital Equipment Corp. President Kenneth Olsen last week promised "drastic" cuts in system prices 13

The quality assurance coordinator of a Boston bank talks about her job 14

A strategic planning company, Cross Information Co., suggests how users can deal with the effects of the AT&T divestiture 15

California legislators have introduced a bill to increase flexibility in prosecuting hackers 16

Interview: Thomas M. Nies, president of Cincom Systems 17

Attendees at the 1984 Insurance and Protection Conference of Financial Institutions were advised to coordinate micro strategy 19

PC World Exposition: Rivalry between IBM and Apple highly evident... New family of IBM-compatible micros, Winchester disk controller among the debuts at the show 22-23

Bottom-up system design is the corporate personal computer networking approach for the '80s, Amy Wohl said recently 24

Mobile robots are being sold to persons for tasks too boring or too dangerous for humans 31

Users at the State University of New York at Albany are designing their own programs 34

An on-line data base is helping a Libbey-Owen-Ford division identify sales opportunities 36

ITT Publishing has reduced DP operating costs by 50% since converting from batch 37

An Atlanta public utility has built a modeling data base that is shared by engineers, systems analysts and planners 40

A building supplier bought a program generator to develop its own accounting application without writing all the code from scratch 42

A CAD system cut the time that Ross Gear spends designing tooling for truck manufacturing 45

Wells Fargo Bank saved \$250,000 by switching to dry COM 47

International Report 25
Managers on the Move 26
Off the Press 22
Calendar 54

NEWS

New Zip+4 rules ease access to discounts

By Jake Kirchner
CW Washington Bureau

WASHINGTON, D.C. — The U.S. Postal Service (USPS) has issued new regulations to make it easier for mass mailers to obtain discounts by using nine-digit Zip Codes. The post office hopes the regulations will greatly increase participation in the new postal automation program.

The Zip+4 program went into effect late last year after several years of controversy. However, the hoped-for success of the program, designed to bolster USPS productivity through greater automation, has been slow in developing, in part because mailers have had problems converting their address files to the longer Zip Codes.

The new regulations, published Jan. 31, should make it easier for firms to qualify for the one-half cent discount for every piece of mail that carries a Zip+4 code while phasing in full conversion to the new numbers, according to USPS and postal experts.

Postal regulations allow mailers to obtain a three cent discount for each piece of first-class mail delivered to a post office in bundles presorted by five-digit Zip Codes. Before the rules were modified, 100% of those letters had to carry a nine-digit code in order to qualify for the additional one-

half cent discount.

But problems with the computerized Zip+4 data base tapes available from USPS and private sources have made it very difficult to convert all files to the Zip+4 program.

These difficulties have not deterred the USPS from expanding its use of new, automated reading and sorting equipment needed for the Zip+4 program.

At a recent USPS Board of Governors meeting, postal officials said the machines are reliably sorting approximately 50% of the mail they handle — far above the break-even point, they said. The board then approved the next phase of equipment procurements, totaling \$450.2 million in machinery.

To prod more mailers to use the longer Zip Codes, the new postal regulations allow comingling of five- and nine-digit Zip Coded mail in presorted bundles of 500 letters. Only the letters with the nine-digit codes will qualify for the extra one-half cent

discount but, on a temporary basis, mailers will not have to separate mail into the two presort categories.

By having to separate the two categories, mailers would have had to forfeit the three cent discount in order to qualify for a one-half cent discount, which understandably limited Zip+4 participation. Under the new rules, until Feb. 1, 1985, at least 85%

of the pieces in the combined batches must have Zip+4 numbers. From then until Oct. 1, 1985, at least 90% must bear those numbers. Thereafter, no comingling will be allowed, and presorted five- and nine-digit Zip Coded mail must be bundled separately. Reacting to these changes,

James E. Pehta, vice-president of List Processing Co., Lombard, Ill., said the rules will bring into the Zip+4 program many of the largest mailers, whose participation "is key to the success of the [Zip+4] program," but who have been "sitting on the fence" because of the problems they have

had in qualifying for the discount.

Pehta, whose firm offers automated Zip Code services, is a member of the USPS Mailers Technical Advisory Committee, which represents private sector views to the agency and which helped to develop the new regulations.

According to Pehta, of the 67 billion pieces of first-class mail delivered in fiscal 1983, 13 billion were presorted.

About 4,000 of the largest mailers in the U.S. produce approximately 40% of the mail, he added.

Pehta predicted that as a result of the new regulations, the amount of mail with nine-digit Zips might hit two billion pieces in the first year, up from an estimated 300 million that could have been expected without the more relaxed regulations.

He also noted that the new regulations contain provisions mailers must follow to certify that their mail meets requirements of the new rules. These will help the USPS track use of the Zip+4 numbers.

The USPS can use this statistical data to support requests for even larger Zip+4 discounts, according to Pehta, who said a three to five cent discount for machine-readable Zip+4 coded mail within four to five years is possible.

The new regulations should make it easier for firms to qualify for the one-half cent discount for every piece of mail that carries a Zip+4 code while phasing in full conversion to the new numbers.

Burroughs releases laser printer

DETROIT — The North American release of the first laser printer for Burroughs Corp. mainframes was announced by the company last week. The B9290-30, a 30 page/min intelligent laser printing system that operates on-line with Burroughs systems, was first introduced in France seven months ago.

The B9290-30 is compatible with Burroughs B2900, B4900, B5900, B6900, B7900 and the recently released B9 mainframes. Running in a continuous print mode, the laser printer reportedly operates under both printer and host system software control.

Burroughs said the printer allows

flexibility in the design of printed forms (printing logotypes and signatures), the placement of data on the forms and the assembly of completed reports. Images are created by a laser diode with a resolution of 57,600 dot/sq in.

The B9290-30 can print on two sides of uncoated, 8 1/2 by 11-in. plain bond paper in either the portrait (standard text) or landscape (columnar) format with no loss of speed and completely collate and stack completed reports in distribution order, the vendor said.

The B9290-30 is priced at \$65,000 from Burroughs, 1 Burroughs Place, Detroit, Mich. 48232.

SUPERSTRUCTURE

Cures COBOL'S Common Code:

Interparagraph GOTOs
ALTER Statements
Altered GOTO Branches
Fall Throughs
Dead Code
PERFORM Range Violations

SUPERSTRUCTURE takes your unstructured COBOL programs and automatically produces structured COBOL programs that are easy to understand and maintain. SUPERSTRUCTURE provides a simple, cost effective alternative to manually rewriting those unstructured programs that are a maintenance headache.

We're so sure we can cure your common code that we'll prove it to you using your programs at your location. Call Marketing Director—SUPERSTRUCTURE today for details.

And soon you'll breathe easier with unstructured COBOL.



Group Operations Incorporated
1110 Vermont Avenue, NW
Washington, DC 20005
(202) 857-6120

Offices in Boston, Chicago, Dallas, Los Angeles and New York

2000 from page 7

The only real solution is to write all new programs to be year 2000-compatible.

Not surprisingly, Schoen has developed a method to do that. The Charmer Correction is a package consisting of two Cobol subroutines that can be inserted into new programs to resolve the problem. The \$996 purchase price also includes an analysis of the problem and a methodology to deal with it.

Programs made to work

"They make programs work right, and they include directions to make sorts simple," he said.

Schoen has done some calculations to show why it makes sense to tackle the problem today. He figures it costs \$300 to modify a program, and there is a minimum of 60 programs per programmer in the average corporate li-

brary. Assuming that half of those programs will need to be modified, that comes to a cost of \$7,500 per programmer if the conversion is left until the last minute.

Based on scans of existing libraries, Schoen has decided that if firms begin implementing the changes now, by the year 2000 less than 2% of their programs will require modification. If they wait just six years to begin the process, the figure balloons to 25% to 30% of their library.

"Why should companies continue to write software that is not year 2000-compatible when it's just as easy to do it the right way now?" he reasons. Not many DP managers have bought his argument so far. He has been escorted out of buildings by security officers more than once, Schoen said.

Schoen's Charmer Enterprises can be reached through P.O. Box 702, Novi, Mich. 48050.

WORLD
ISSUES

COMPUTERWORLD

Site Map

Search

Contact Us

Subscribe

News & Features

Resource Center

IT Careers

Computerworld, Inc.

News

- Latest news headlines
- Shark Tank
- CW Minute (audio)
- Continued from print
- Computerworld Newspaper

Features

- Managing
- Year 2000
- Field Report
- In Depth

Viewpoint

- Maryfran Johnson
- More columnists

Latest publications

- Most recent publications

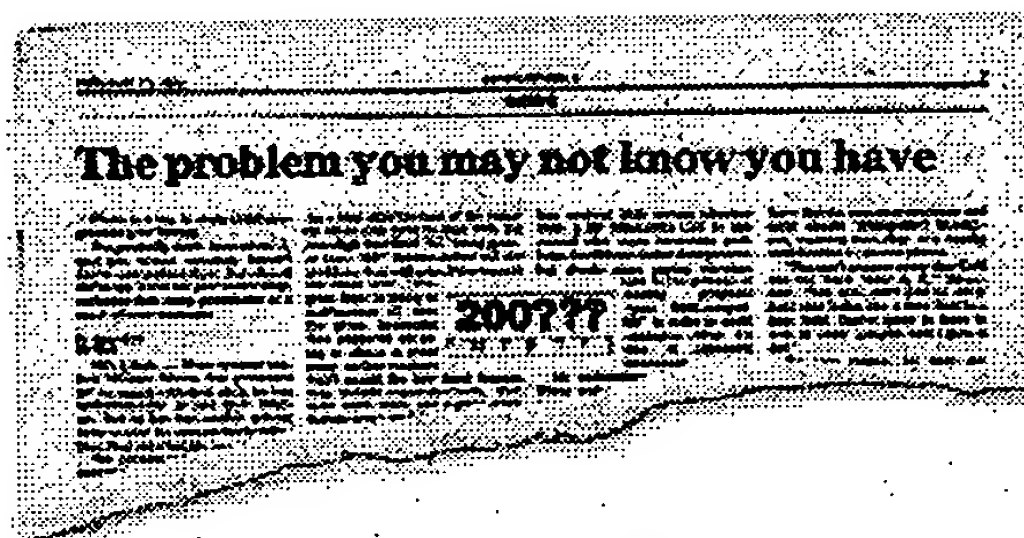
A Y2K pioneer seeks (and deserves) recognition

Bill Schoen was passionate about the problem long before the world was ready to listen

By Paul Gillin

Opinion, Aug. 3, 1998 It was a frigid Wednesday in February 1984, and I was stumped for a story idea. The software business was dead that time of year — not good news to a guy who wrote about software for *Computerworld*.

Then the phone rang. On the other end was a guy from Detroit who had an idea that was so offbeat, so screwy, that I thought it just might make my editors happy. It was good for a laugh, at least.



Read the Feb. 13, 1984 column

Bill Schoen had this wacky idea that a lot of computers — I mean a lot of them — would stop working on New Year's Day 2000. Schoen wasn't just curious about this problem, he was passionate about it. He had even formed a one-man consulting company to try to spread the gospel. But no one, he said, wanted to listen.

I listened. Heck, I was desperate. I interviewed Schoen, talked to one of his clients and wrote it up. My editors put the story on page 7 of the Feb. 13, 1984, issue. It was the first article to appear in a major publication about the year 2000 problem (see [the original story](#).)

I spoke to Bill Schoen again the other day. Now 51, he's still in Detroit, still programming and a little amazed about how prophetic his crusade was. Though he wishes he could have had a piece of the five-figure speaking fees that top year 2000 consultants make, what Schoen really wants today is a little recognition.

Schoen first stumbled on the year 2000 problem in 1983 while programming in the basement of a Big Three automaker. It wasn't rocket science; data processing people had known about the risk since the 1970s. It's just that no one thought their code would last

Schoen thought it might, and a little analysis of his employer's code library proved to him that software had considerably more staying power than many people thought. If business had started coding for the millennium in 1983, "95% of the problems wouldn't be there today," he says wistfully.

For Schoen, the year 2000 problem became a mission. He coded up a little Cobol routine on his Commodore 64 that solved the problem. He named his company Charmar Enterprises and created the Charmar Correction, a cure for "the serious problem ignored by the entire data processing community."

He was an army of one. Booted out of the CIO's office in some of the biggest companies in America, Schoen landed only two sales for the Charmar Correction. He folded Charmar's tent in 1984 and went back to programming. "I've never been able to rise above the level of working stiff," he says.

AHEAD OF HIS TIME

Schoen's story is about missed opportunity and being too far ahead of the curve. The New York Times called him in 1988 to ask about the year 2000 problem, but Schoen was sick of rejection and asked the reporter not to quote him. His efforts to launch a year 2000 consultancy last year fizzled when the client's CEO died.

Now he's got a Web site that tells a little of the story of the guy who had an answer long before anyone else was asking the question.

Visit Bill Schoen's Web site. Send him an E-mail. This guy was really on to something.

Gillin is editor of Computerworld. His Internet address is paul_gillin@cw.com.

[NEWS & FEATURES](#) | [RESOURCE CENTER](#) | [IT CAREERS](#) | [COMPUTERWORLD, INC.](#)
[Contact Us](#) | [Search](#) | [Site Map](#) | [Subscribe](#)

COMPUTERWORLD

Copyright © 2000 Computerworld, Inc. All rights reserved. [Legal notices and trademark attributions](#)



 [Back to top](#)

The problem you may not know you have

There is a bug in every Cobol program in your library.

You probably don't know about it, and you almost certainly haven't had to worry about it yet. But when it shows up, it will hit your entire shop, reducing data entry procedures to a mush of error messages.

By Paul Gillin
CW staff

NOVI, Mich. – When systems analyst William Schoen has presented DP managers with that pitch, he has understandably piqued their interest. But he has had trouble getting them to take his case seriously when they find out what the bug is.

The problem is the year 2000. The turn of the calendar presents a procedural issue that is acknowledged occasionally in trade conference jokes and DP shop banter but has attracted little serious attention in the industry.

The root of the issue is the industrywide standard of using two-digit year date fields in Cobol programs. Error-checking procedures typically rely upon dates proceeding sequentially, with one year's date being greater than that of previous years.

However, programs will be thrown for a loop after the turn of the century when they have to cope with the two-digit date field "00" being greater than "99." Schoen ticked off the problems that will arise if the issue is not dealt with: "Program logic is going to malfunction all over the place. Sequential data processes are going to abend. A great many on-line modules won't accept the new dates because they include sequence checks. Most sorts won't work, and a great many literals won't work."

Perhaps understandably, the issue has received little serious attention from a DP community that is concerned with more immediate problems. But Schoen claims data processing should start paying attention now to the problem of making programs "year 2000-compatible" in order to avoid headaches when the time of reckoning draws near.

He rationalizes that most large firms maintain a library of thousands of Cobol programs, most of which have at least one date field. In addition, many of those programs have literals, sequence processes and error checks interspersed throughout, meaning that they will require modifications in several places.

"You can't assume every date field has the word 'date' in it," Schoen said. "You also can't assume every field that looks like a date field is a date field. You're going to have to look at every program and figure it out."

For that reason, he said, the "black box" approach of simply running every program through a modification routine is impractical. Some date fields are bound to be missed. The only real solution is to write all new programs to be year 2000-compatible.

Not surprisingly, Schoen has developed a method to do that. The Charmar Correction is a package consisting of two Cobol subroutines that can be inserted into new programs to resolve the problem. The \$995 purchase price also includes an analysis of the problem and a methodology to deal with it.

Programs made to work

"They make programs work right, and they include directions to make sorts simple," he said.

Schoen has done some calculations to show why it makes it makes sense to tackle the problem today. He figures it costs \$300 to modify a program, and there is a minimum of 50 programs per programmer in the average corporate library. Assuming that half of those programs will need to be modified, that comes to a cost of \$7,500 per programmer if the conversion is left to the last minute.

Based on scans of existing libraries, Schoen has decided that if firms begin implementing the changes now, by the year 2000 less than 2% of their programs will require modification. If they wait for just six years to begin the process, the figure balloons to 25% to 30% of their library.

"Why should companies continue to write software that is not year 2000-compatible when it's just as easy to do it the right way now?" he reasons. Not many DP managers have bought this argument so far. He has been escorted out of buildings by security officers more than once, Schoen said.

Manager gets 16-year jump on Cobol bug

TOLEDO, Ohio – Although he agrees that making Cobol programs "year 2000-compatible" is not an issue of immediate concern, a DP manager here has elected to become the first user of a set of subroutines designed to make the transition to the new millennium a smooth one.

Michael Ehrick, director of computer services at Libbey-Owens-Ford, Inc., became interested in preparing Cobol programs for the inevitable march of time after he received a direct mail advertisement from William Schoen, a systems analyst based in Novi, Mich.

Schoen's Charmar Enterprises, Inc. markets a product that addresses the problems that may emerge when the year 2000 arrives and plays havoc with date fields in existing Cobol programs (see related story). His product is "a subroutine that does some elegant arithmetic routines that let you compare date fields with two-digit years in such a way that the year '00' is greater than the year '99,'" Ehrick said. Schoen also offers an approach to sorting so that the same end is achieved, according to Ehrick.

Ehrick put Schoen in contact with Libbey-Owens-Ford's manager of application development. The company expects to begin including the subroutine in new applications as soon as the next

The subroutines are "nothing an advanced computer science major couldn't do," Ehrick said. But he added, "a lot of creative thought went into this, along with a lot of dogmatism."

Copyright 1984 Computerworld Inc., Framingham MA.
Reprinted by permission of Computerworld.

*13

35

A Vision of Global Virtual Computing
A Special Section High-Assurance Systems



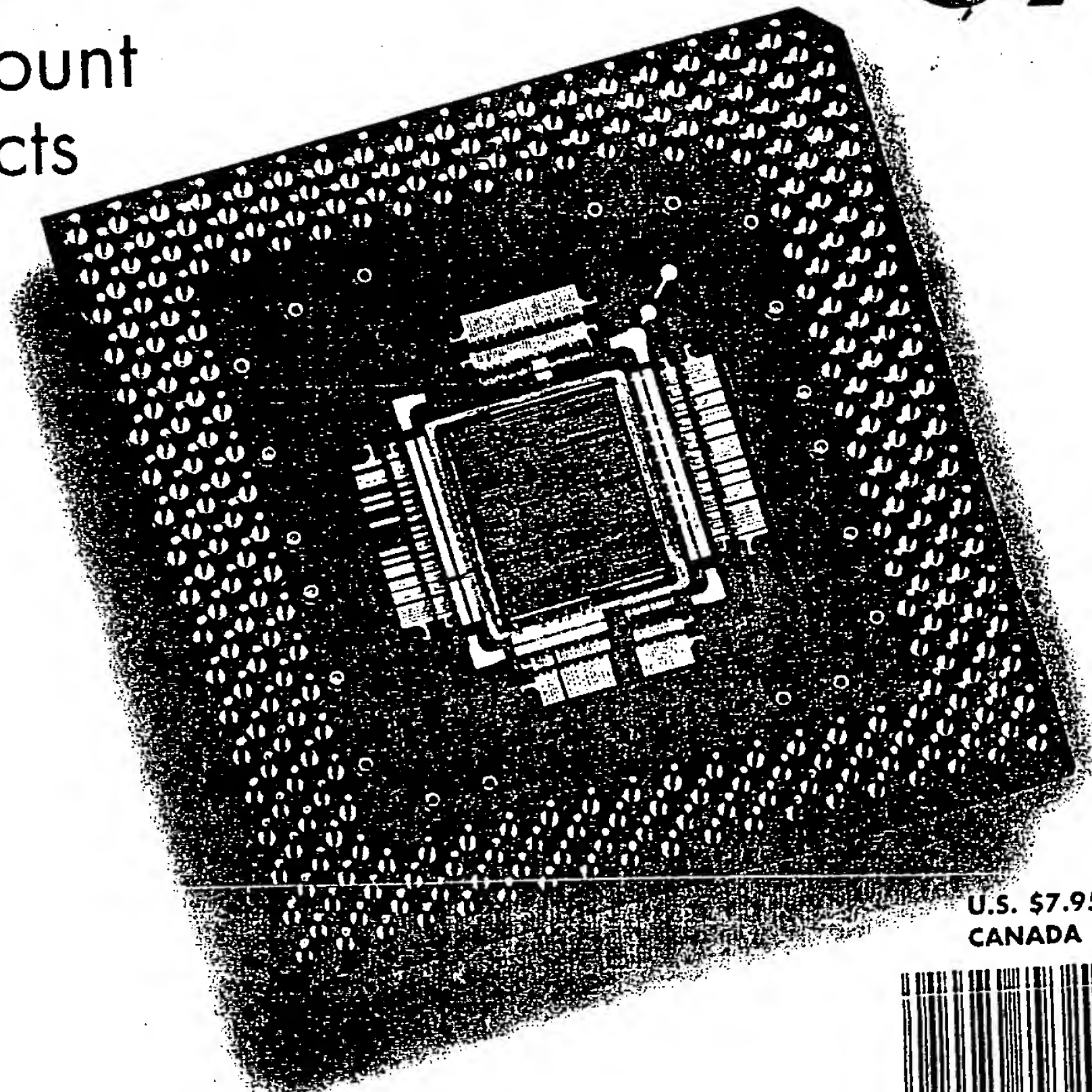
COMMUNICATIONS

of the ACM

January 1997—Volume 40, Number 1

How Intel Built MMX Technology

An inside account
by the architects



***** 5-DIGIT 47907
1292242 11 EXPIRE 9701 101 1 01
JOHN R. RICE
DEPT OF COMP SCI
PURDUE UNIVERSITY
WEST LAFAYETTE IN 47907
DEC15 012
034

U.S. \$7.95
CANADA \$10.95



0 74470 79329 8

January 1997

Table of Contents

Cover Story

- 24 Intel MMX for Multimedia PCs *Alex Peleg, Sam Wilkie, and Uri Weiser*

Articles

- 39 The Legion Vision of a Worldwide Virtual Computer *Andrew S. Grimshaw, Wm. A. Wulf, and the Legion Team*

- 46 Computer Virus—Antivirus Coevolution *Carey N. Nienberg*

- 52 Artificial Intelligence and Virtual Organizations *Daniel E. O'Leary, Daniel Kuokka, and Robert Plant*

- 60 Ethics Online *Deborah G. Johnson*

High-Assurance Systems

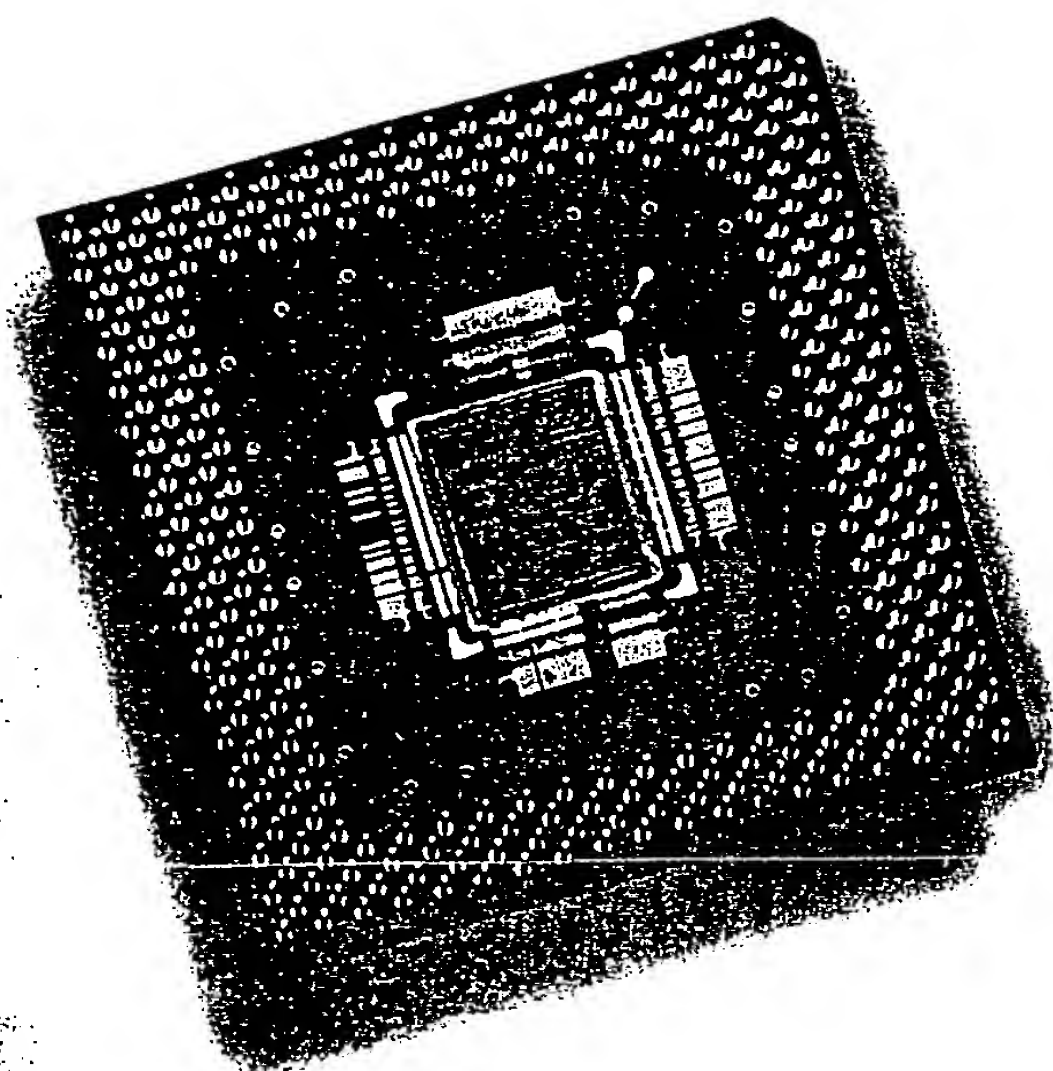
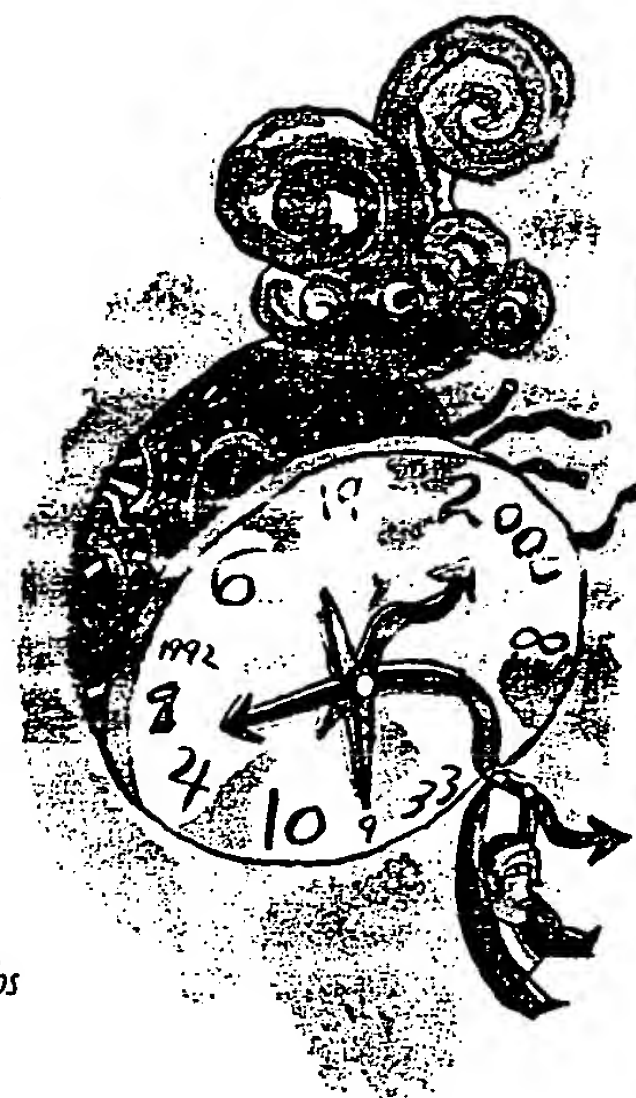
- 67 Introduction *Sourav Bhattacharya, Akira Onoma, and Farokh Bastani*
Guest Editors

- 68 Adaptive Recovery for Mobile Environments *Nuno Neves and W. Kent Fuchs*

- 75 Fault-Tolerant Real-Time Objects *K.H. (Kane) Kim and Chittur Subbaraman*

- 83 Parallel Evaluation of Software Architecture Specifications
Jeffrey J.P. Tsai, Bing Li, and Eric Y.T. Juan

- 87 Mechanisms for Detecting and Handling Timing Errors *David B. Stewart*
and Pradeep K. Khosla



Columns

- 11 From Washington If It Grows, Tax It
Neil Munro

- 15 Practical Programmer The Next Date
Crisis and the Ones After That
Robert L. Glass

- 20 On Site Tomorrow's Library Today
Robert Fox

- 138 Inside Risks Cryptography, Security,
and the Future
Bruce Schneier

Departments

- 9 News Track
19 ACM97 Speakers Corner
22 Forum
94 Technical Opinion
95 Calendar of Events
97 Calls for Participation
104 Index for Advertisers
101 Career Opportunities

COMMUNICATIONS

OF THE ACM

A monthly publication of the ACM

Publications Office
ACM, 1515 Broadway,
New York, New York 10036-5701 USA
(212) 869-7440 FAX: (212) 869-0481

Editorial Information
email: editor@acm.org
Change of address: acmcoa@acm.org
Calendar items: calendar@acm.org
Member Services Information:
acmhelp@acm.org

Executive Editor: Diane Crawford
Managing Editor: Thomas E. Lambert
Senior Editor: Andrew Rosenbloom
Associate Editor: Robert Fox
Editorial Assistant: Helen Pacheco
Copyright: Deborah Cotton

Contributing Editors
John Perry Barlow; Robert L. Glass;
Seymour Goodman; Brock N. Meeks;
Neil Munro; Peter G. Neumann; Larry Press;
Roy Rada; Pamela Samuelson; Elliot Soloway

Art Director: Caren Rosenblatt
Production Manager: Lynn D'Addesio

Advertising Director: Walter Andrzejewski
Advertising Assistant: Silvia Castex
Advertising Coordinator: Michele Bianchi

Display Advertising Representatives

Northeast/Midatlantic/Midwest/Foreign:
Walter Andrzejewski
(212) 626-0685; Fax: (212) 869-0481
email: andrzejewski@acm.org

Northeast:
The Summit Group
(201) 442-3998; Fax: (908) 876-9332
email: hersh@ibm.net

Southeast:
Ray Rickles and Company
(770) 664-4567; Fax: (770) 740-1399

Midwest:
Bart Engels
(847) 2854-6050; Fax: (847) 854-8183
email: engels1@aol.com

West:
Marshall Rubin and Associates
(818) 995-8828; Fax: (818) 995-6366
email: mrubin@westworld.com

Classified Advertising

ACM Advertising Department,
1515 Broadway, New York, NY 10036-5701.
(212) 869-7440; Fax: (212) 869-0481.
email: acm-advertising@acm.org

Communications of the ACM

(ISSN 0001-0782) is published monthly by the
ACM, 1515 Broadway, New York, NY
10036-5701. Periodicals postage paid at
New York, NY 10001, and other mailing offices.
POSTMASTER: Please send address changes to
Communications of the ACM, 1515 Broadway,
New York, NY 10036-5701 USA.



Printed in the
U.S.A.

Editorial Pointers

IF all goes as scheduled, Intel will introduce its highly anticipated MMX (multimedia extensions) technology with the Pentium processor later this month. Thus begins one of the most heated races among chip designers worldwide: to create or add multimedia functions that allow for the speedy use of 3D graphics, audio, video, and enhanced communications. Intel may be the biggest player, but the competition is indeed fierce with several smaller firms perhaps better equipped to design special media processors than the mighty, multipurpose chip giant. Indeed, MMX marks the first major change Intel has made to its chip instruction set technology in over a decade.

This month's cover story is an inside, in-depth account of Intel's creation of MMX technology as told by three of its principal architects: Alex Peleg, Sam Wilkie, and Uri Weiser. They explain how MMX—and its 57 new instructions and other enhancements—hikes system performance dramatically. Their report explores the many features of the technology using simple examples as guidelines.

In other news this month, we learn from Andrew Grimshaw and William Wulf of a fascinating new concept in virtual world computing called "Legion." Carey Nachenberg traces the current causes, effects, and remedies of computer viruses and explores what we can expect in the future. Deborah Johnson interprets the controversial issues related to ethical behavior online. And O'Leary, Kuokka, and Plant detail the use of artificial intelligence, particularly software agents, to facilitate virtual organizations. Finally, we present a special section on the latest advances in high-assurance systems engineering spearheaded by guest editors Sourav Bhattacharya, Akira Onoma, and Farokh Bastani.

WITH this issue we begin taking steps in a new editorial direction to enhance our own performance as a vehicle for communicating the latest industry developments and technology innovations. Guiding us on this journey is a new *Communications* advisory board comprised of 27 noted experts representing a strong mix of industry, academia, and research. They will help pinpoint newsworthy topics, anticipate industry trends, and identify authors to pursue.

We will also be introducing new features and columns in coming issues. We begin this month with a new column called "On Site" designed to provide a forum for first-person experiences with new applications, unique solutions to common technical problems, or innovative new uses for existing tools and techniques. Associate Editor Robert Fox starts us off with a report of his experiences visiting two of the best examples of electronic public libraries. He wonders how the general public will respond to the digitization of its local library facilities and we think his findings will surprise you.

Diane Crawford

Executive Editor

COMING NEXT MONTH: A special issue celebrating
our hopes for the next 50 years of computing.

The Next Date Crisis and the Ones After That

Robert L. Glass

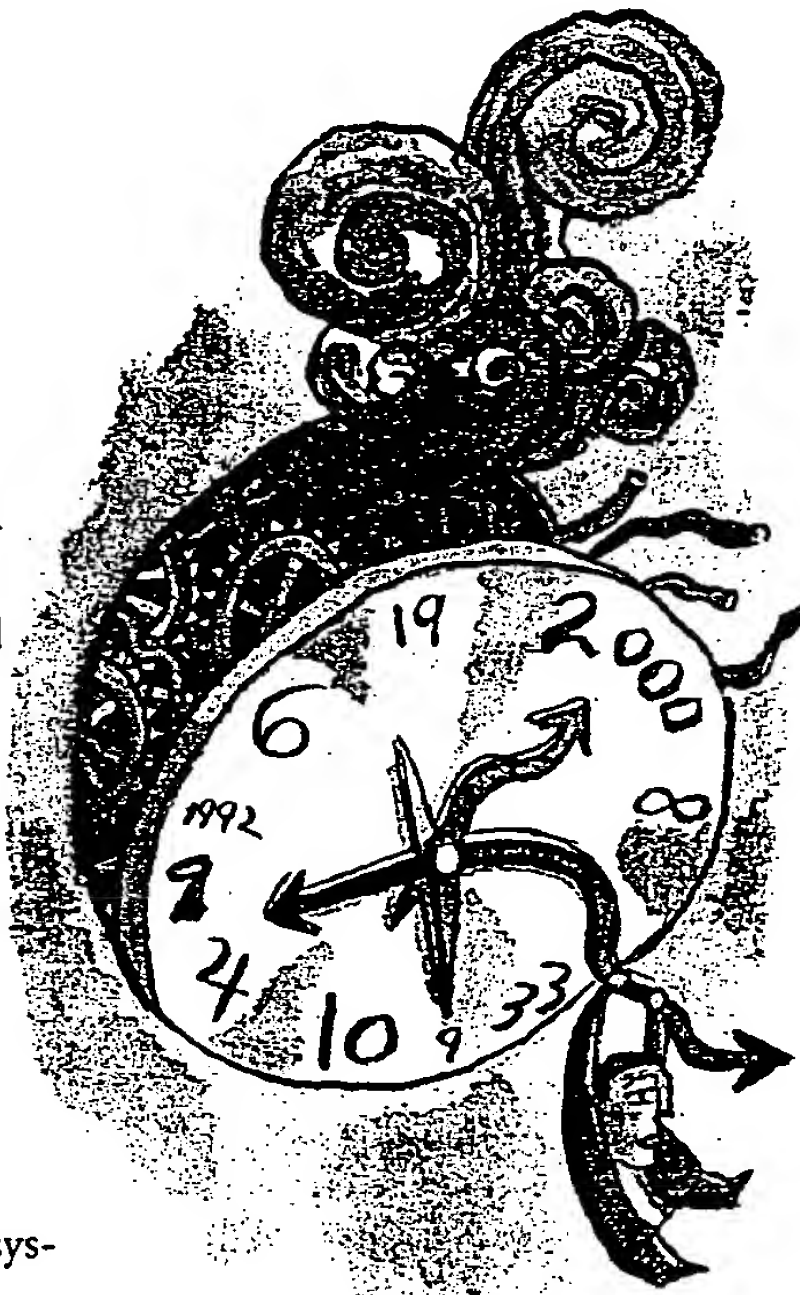
Reality is the murder of a beautiful theory by a gang of ugly facts.

I'm sure you already know about how all information systems will go berserk when the year 2000 comes, the so-called "date crisis" everyone is talking about.

This column is about a date crisis. But not *that* date crisis. I want to talk about the one that comes *after* the one everyone knows about.

First, to make sure we're all on the same page, let me describe the date crisis everyone already knows about. Dates in most information systems are stored in the form MM/DD/YY. That is, two digits are used for each month, day, and year. That's not a problem for months or days, of course.

But years? Big problem. Even though most of us say "the 90s" and in many other ways use years as if two digits were enough, we're approaching 2000, when two digits simply aren't sufficient to do the job. Lots of information systems do arithmetic on dates (e.g., how old is a person whose birthdate is 02/03/32?). Information-system arithmetic will



fail come the year 2000. (By the way, here's the answer to the question—in 1999, 67. In 2000—"minus" 32.)

Remember, this column is *not* about that date crisis. I'm assuming nearly everyone has read and listened to some of the thousands of words spilled over the last several years on this topic. And I'm also assuming nearly everyone is aware the crisis is almost inevitable because hardly any

companies have allocated and spent the resources necessary to fix the problem. There *will* be a crisis at the turn of the century, although it probably won't be the World War III some consultants are hyping it to be.

What, then, is the date crisis I want to discuss?

No matter how—or if—an enterprise solves its year-2000 date crisis, another one is coming. And the form of the solution chosen for this particular crisis may determine when the next crisis will occur.

First of all, let's state the obvious: There is no permanent solution to the date crisis. Whereas the months of the year recycle after 12, and days of the months recycle after about 30, years just keep marching on toward infinity. And computers with their finite word lengths cannot hold them forever. That explains part, but not all of, the next date crisis.

Let's say you decide to employ what to most people in 1997 is the obvious solution to this date crisis—the four-digit year. You change all your data files and data declarations to accommodate a couple more digits per year, test all the software affected by the change, and sit back and sigh the sigh of satisfied relief: one more crisis solved. Not so fast. Your solution is certainly valid for the foreseeable future, of course. But it will fail come the year 10000. As surely—and in the

same way—as the 2000 date crisis.

It's hard to get excited about a future 8,000 years ahead. For one thing, you're probably saying, "What are the odds that information systems as such will be around that long?" Except that's the same kind of thinking that got us into the year 2000 mess. No one believed, back in 1975, the information systems of that day would last another 25 years.

Still, no one is *really* going to worry about the four-digit solution. After all, the five-digit solution—or any other solution you contemplate—will also have its very own day of doom associated with it. How far off is it necessary to postpone this problem?

There are other, more sinister, forms the next date crisis can take. In fact, the Unix operating system

the procedure decide whether the date 56 represents 1956 or 2056? The answer is usually a bad one, going like this: Pick an arbitrary date that will be accurate today and use it as a watershed date. For example, any year less than 50 is in the next century, and any greater than or equal to 50 is in the previous one. Ergo, 56 = 1956. Problem solved?

Today, perhaps. But what happens as we approach 2050? The solution comes unglued, just as badly as the year-2000 problem. Or perhaps worse, because the solution is local to the enterprise (other companies might have chosen 60 or 70 instead of 50) and thus there will be fewer cries of warning next time. But no one would use this solution, you may say. Not true. A consultant writing in a leading journal proposed it

dates have always been kept in decimal form in our binary (and, before that, decimal) computers, we assume blindly it must always be so. In doing so we neglect another form of data that computers use—binary. Suppose we store the year in binary rather than decimal form?

Most years today, being in YY form, occupy 16 bits (two eight-bit bytes). (This is not universally true. For example, Cobol has a numeric form called "packed decimal" that alternatively allows digits to be stored in four-bit form, so with packed decimal, a year would require 8 bits.) If a number is stored in binary form in 16 bits, it can be as large as 65,536 (64K). That is, using the same two bytes that YY occupies a year of up to 65536 could be stored without increasing the memory size

NO ONE BELIEVED, BACK IN 1975, the information systems of that day would last another 25 years.

apparently has its own built-in date crisis ticking away like a time bomb, and it's difficult to predict precisely when this particular bomb is going to explode.

But for now let's continue to talk about IS applications. Most enterprises, as I've said, will employ the four-digit-year approach. Some are troubled by the expansion of databases if all the two-digit years get converted to a bigger number.

Because of that, some companies are trying another solution: Keeping the two-digit dates on file, but having a procedure in the program to convert each date before its four-digit equivalent is used. So far, so good. This solution contains a serious problem, however. For instance, how does

within the last 10 months. How many companies are following that very bad advice?

The point is the date crisis is a "pay me now, *and* pay me later" kind of problem. There *will* be another date crisis sometime in the future, and the decisions enterprise IS people make today will determine when that next crisis will be.

At this point, you may find yourself sympathetic with those concerned about the increase in storage consumed by the four-or-more digit year. Given that the "greater than 50" solution is a bad one, is there any other solution that will prevent database expansion?

Turns out there is one and hardly anyone is talking about it. Because

in the database. In order to use this solution, however, the database would have to be rewritten with years in binary form, the program logic would have to declare years as binary (there's a form of *computational* in Cobol that will do the trick), and all the program date logic would need to be checked to make sure nothing more is affected by the change (it is possible the logic may not need to change, except for human-readable outputs). (Note this solution is not valid for packed decimal dates, discussed earlier. Whereas 16 bits gives 64K, 8 bits only gives 256, insufficient to store years.)

So in this "pay me now *and* pay me later" game, you pay your money and you take your chances. You can

have another date crisis in the next century (around 2050 or so), at 10000, or at 65536. The choice is yours. There are pros and cons to each approach. Only the technologist in charge of an enterprise's suite of applications can make the best decision, because there is no universal best one. Although the four-digit solution comes close.

And what about the Unix-unique date crisis mentioned earlier?

Here's my understanding of what happens in Unix. Bear in mind that I am *not* a Unix guru. It may very well be that I'm wrong. I hope I am. But if I am correct there may be a date crisis for applications on Unix systems that could become the mother of all date crises.

Unix has a device called "Unix time," the measurement of time in seconds from a base time—the first day of 1970. Unix time, by 1997, has gotten quite large. There are roughly 31 million seconds in a year. In 27

years, we have piled up 27×31 million, or 837 million seconds. The number is constantly increasing—by, for example, 300 or more since you started reading this column.

Numbers in computer speak are finite. No matter how large a fixed space you set aside, as numbers charge madly toward infinity they will outgrow their space. Will Unix time outgrow its space? Inevitably. The question is, when?

The answer is hardware dependent. A computer's word length probably determines the date on which Unix time will overflow. (Go from the biggest number back to 0 when the capacity is exceeded). Let's make a typical case assumption and say a computer has a 32-bit word. If my arithmetic is correct, Unix time will overflow around 2035. Sooner than even the worst of the date crisis solutions contemplated, and even sooner for smaller-word processors.

This is one of those "surely I'm

wrong" kinds of findings. Surely the designers of Unix anticipated such a problem and have provided for it. Does anyone in my reading audience know? If so, tell me and I'll pass the answer on in a later column. In a way, it sounds like I'm crying "fire" in a crowded theater. If there is no fire, it's important I be set straight. But if I'm right, it's time to begin talking about what Unix and its application programmers must do.

So we see that there's yet another software date crisis (YASDC) coming. That if you're using Unix you may have very little choice about the date it arrives. That for all IS applications, using Unix or not, there's no choice as to *whether* there will be YASDC. Your only choice is *when* to schedule it. \square

ROBERT GLASS is the publisher of the Software Practitioner newsletter and editor of Elsevier's Journal of Systems and Software. He welcomes feedback: 1416 Sare Rd., Bloomington, IN 47401.

Communications of the ACM

March 1997

Collaborative Filtering (Recommendation Systems)

Also in this issue:

Robots/Telecommunications
and the Impact on Business
in the Real World

essential

learn about this new area of
filtering information using
people and computers

effective

Read the decision makers interested in
the competitive edge

advertising Close: January 20, 1997 +1-212-626-0685 acm-advertising@acm.org

VOLUME 37 - NUMBER 08 - AUGUST 1994

36

IBM Technical Disclosure Bulletin is published monthly by International Business Machines Corporation, Stamford, CT 06904. Officers: Louis V. Gerstner, Jr., Chairman of the Board & CEO; Paul J. Rizzo, Vice Chairman of the Board; F.W. Zuckerman, IBM Vice President & Treasurer; John E. Hickey, Assistant General Counsel & Corporate Secretary; Vynetta W. Ross, Editor.

Inquiries should be directed to Intellectual Property Law, International Business Machines Corporation, Armonk, New York 10504.

©Copyright International Business Machines Corporation 1994. Printed in the U.S.A. The publication of these technical disclosures does not constitute a grant of any license under patent.

NOTICES

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in the publication at any time.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not grant any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, Stamford, Connecticut 06904.

Copyright License

IBM Corporation, copyright owners of the IBM Technical Disclosure Bulletin, and any other copyright owners of articles published in this bulletin hereby give consent for copies to be made of articles herein which are for personal use, or for personal or internal use.

For copying beyond the permission given above, or permitted by Sections 107 or 108 of the US Copyright Law, this consent does not extend to other kinds of copying such as copying for general distribution for advertising or promotional purposes, or for creating new collective works for resale.

PRINTED AND PUBLISHED IN THE UNITED STATES OF AMERICA

© IBM Corp.

Method of Sorting Dates and Time Allowing for Wrapping

Disclosed is a method of sorting dates, times, or other data which may wrap the counter. The method does not require specification of an arbitrary window which is assumed to contain the times, but rather uses the data and the information that the numbers are all given modulo something in order to determine the correct ordering. Specific applications include the problems of sorting two-digit years which may cross a century boundary, months which may encompass the end of the year, and times which encompass midnight. The algorithm applies to sorting all cyclical groups of numbers; the sorted numbers need not represent time.

The problem solved may be illustrated with dates in the neighborhood of the year 2000. The years labelled '03, '05, '02, '99, '96, would properly be sorted chronologically as '96, '99, '02, '03, '05. A conventional sorting routine would sort them as '02, '03, '05, '96, '99. To obtain the correct sorting, a traditional approach is to pick an arbitrary year and assume that all dates are after that year, pre-append the appropriate century, and then use a conventional sorting algorithm. For example, one might assume that all the specified dates are after the year 1975. So '96 must mean 1996 and '03 must mean 2003. The disclosed method avoids this arbitrary designation of a date. Instead, it examines the data, taking into account the possibility of wrapping, finds the largest gap, and then shifts the sort so that the number after the largest gap comes first. For the dates '02, '03, '05, '96, '99, the successive differences are 1, 2, 91, 3, and 3 years. The second difference of 3 is found by using the understanding that the original numbers are given modulo 100, i.e. using circular subtraction. Since the third difference, 91, is the largest, the fourth date, '96, should come first, and the correct sorting is then '96, '99, '02, '03, '05.

Let MAXNUM be the largest possible number representable, that is, one less than the modulo base. For example, the largest year representable with two digits is '99. We wish to sort the array A(i), taking into account the fact that the numbers A(i) may wrap. The method is

1. Sort the numbers A(i) using a conventional sorting routine.
2. Find the largest gap in the sorted numbers, using circular subtraction. For N numbers, this is illustrated with the pseudocode

$$\text{DIFFMAX} = A(1) + \text{MAXNUM} + 1 - A(N)$$

$$\text{STARTI} = 1$$

do I = 1 to N-1

$$\text{DIFF} = A(I+1) - A(I)$$

if DIFF > DIFFMAX then do

$$\text{STARTI} = I + 1$$

$$\text{DIFFMAX} = \text{DIFF}$$

end

end

The largest gap, of size DIFFMAX, occurs before A(STARTI). In case of ties, this algorithm chooses the first occurrence of the largest gap.

3. Shift the sorted numbers so that the number after the largest gap comes first. That is

$J = \text{STARTI}$

do $I = 1$ to N

$B(I) = A(J)$

$J = J + 1$

if $J > N$ then $J = 1$

end

The array $B(i)$ now contains the numbers, sorted by what is likely to be the correct order when the possible wrapping is taken into account. The array $B(i)$ is introduced only for illustrative purposes. Other means may be used to shift the array so that the first element becomes what was $A(\text{STARTI})$ and the circular order is maintained.

This algorithm will produce proper orderings as long as the numbers are reasonably close together in the circular sense. It has the advantage that the algorithm itself will never become out of date. Whether it is appropriate to use depends on the anticipated scatter of the circular data. Years do not typically have a great deal of scatter, which is why notation which drops the century has come to be employed. The algorithm may be expected to work well with such data. The key requirement is that the numbers fall into some reasonable span. A person contemplating sorting the dates '05, '95, '99, and '01 would probably assume that '99 comes before '01. How would they know? They would know of the possibility of wrapping, and guess that wrapping did occur for the little numbers because with that assumption the total span of years is much smaller. The above algorithm accomplishes this minimizing of the total span of years.

Method of Sorting Dates and Time Allowing for Wrapping

Disclosed is a method of sorting dates, times, or other data which may wrap the counter. The method does not require specification of an arbitrary window which is assumed to contain the times, but rather uses the data and the information that the numbers are all given modulo something in order to determine the correct ordering. Specific applications include the problems of sorting two-digit years which may cross a century boundary, months which may encompass the end of the year, and times which encompass midnight. The algorithm applies to sorting all cyclical groups of numbers; the sorted numbers need not represent time.

The problem solved may be illustrated with dates in the neighborhood of the year 2000. The years labelled '03, '05, '02, '99, '96, would properly be sorted chronologically as '96, '99, '02, '03, '05. A conventional sorting routine would sort them as '02, '03, '05, '96, '99. To obtain the correct sorting, a traditional approach is to pick an arbitrary year and assume that all dates are after that year, pre-append the appropriate century, and then use a conventional sorting algorithm. For example, one might assume that all the specified dates are after the year 1975. So '96 must mean 1996 and '03 must mean 2003. The disclosed method avoids this arbitrary designation of a date. Instead, it examines the data, taking into account the possibility of wrapping, finds the largest gap, and then shifts the sort so that the number after the largest gap comes first. For the dates '02, '03, '05, '96, '99, the successive differences are 1, 2, 91, 3, and 3 years. The second difference of 3 is found by using the understanding that the original numbers are given modulo 100, i.e. using circular subtraction. Since the third difference, 91, is the largest, the fourth date, '96, should come first, and the correct sorting is then '96, '99, '02, '03, '05.

Let MAXNUM be the largest possible number representable, that is, one less than the modulo base. For example, the largest year representable with two digits is '99. We wish to sort the array A(i), taking into account the fact that the numbers A(i) may wrap. The method is

1. Sort the numbers A(i) using a conventional sorting routine.
2. Find the largest gap in the sorted numbers, using circular subtraction. For N numbers, this is illustrated with the pseudocode

```
DIFFMAX = A(1) + MAXNUM + 1 - A(N)
```

```
STARTI = 1
```

```
do I=1 to N-1
```

```
DIFF = A(I+1) - A(I)
```

```
if DIFF > DIFFMAX then do
```

```
STARTI = I + 1
```

```
DIFFMAX = DIFF
```

```
end
```

```
end
```

The largest gap, of size DIFFMAX, occurs before A(STARTI). In case of ties, this algorithm chooses the first occurrence of the largest gap.

3. Shift the sorted numbers so that the number after the largest gap comes first. That is

```
J = STARTI
```

```
do I=1 to N
```

```
B(I) = A(J)
```

```
J = J + 1
```

```
if J > N then J = 1
```

```
end
```

The array B(i) now contains the numbers, sorted by what is likely to be the correct order when the possible wrapping is taken into account. The array B(i) is introduced only for illustrative purposes. Other means may be used to shift the array so that the first element becomes what was A(STARTI) and the circular order is maintained.

This algorithm will produce proper orderings as long as the numbers are reasonably close together in the circular sense. It has the advantage that the algorithm itself will never become out of date. Whether it is appropriate to use depends on the anticipated scatter of the circular data. Years do not typically have a great deal of scatter, which is why notation which drops the century has come to be employed. The algorithm may be expected to work well with such data. The key requirement is that the numbers fall into some reasonable span. A person contemplating sorting the dates '05, '95, '99, and '01 would probably assume that '99 comes before '01. How would they know? They would know of the possibility of wrapping, and guess that wrapping did occur for the little numbers because with that assumption the total span of years is much smaller. The above algorithm accomplishes this minimizing of the total span of years.

Proceedings of the

(37)

18th International Conference on Software Engineering

March 25–29, 1996

Berlin, Germany

Sponsored by

The IEEE Computer Society Technical Council on Software Engineering

The Association for Computing Machinery (SIGSOFT)

Gesellschaft für Informatik



IEEE Computer Society Press
Los Alamitos, California

Washington

• Brussels •

Tokyo

MATH
001.6425
In 8f
18th
1996
Set 2



IEEE Computer Society Press
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264

Copyright © 1996 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved.

Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. Libraries may photocopy beyond the limits of US copyright law, for private use of patrons, those articles in this volume that carry a code at the bottom of the first page, provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Other copying, reprint, or republication requests should be addressed to: IEEE Copyrights Manager, IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interests of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society Press, or the Institute of Electrical and Electronics Engineers, Inc.

IEEE Computer Society Press Order Number PR07246

ISBN 0-8186-7246-3

ISSN 0270-5257

ACM Order Number 592960

IEEE Order Plan Catalog Number 96CB35918

IEEE Order Plan ISBN 0-8186-7247-1

Microfiche ISBN 0-8186-7248-X

Additional copies may be ordered from:

IEEE Computer Society Press
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264
Tel: +1-714-821-8380
Fax: +1-714-821-4641
Email: cs.books@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: +1-908-981-1393
Fax: +1-908-981-9667
mis.custserv@computer.org

IEEE Computer Society
13, Avenue de l'Aquilon
B-1200 Brussels
BELGIUM
Tel: +32-2-770-2198
Fax: +32-2-770-8505
euro.ofc@computer.org

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama
Minato-ku, Tokyo 107
JAPAN
Tel: +81-3-3408-3118
Fax: +81-3-3408-3553
tokyo.ofc@computer.org

Additional copies may be ordered prepaid from:

ACM Order Dept.
P.O. Box 12114
Church Street Station
New York, NY 10257

Phone: 1-800-342-6626 (U.S. and Canada)
1-212-626-0500 (all other countries)
Fax: 1-212-944-1318
E-mail: ACMPUBS@ACM.ORG

Editorial production by Mary E. Kavanaugh / Cover production by Joseph Daigle
Printed in the United States of America by KNI Inc.



The Institute of Electrical and Electronics Engineers, Inc.

Table of Contents

<i>Foreword</i>	xi
<i>Dedication</i>	xiv
<i>Organizing Committees</i>	xvi
 Session 1: Keynote Address	
Speaker: Tom DeMarco – <i>The Atlantic Systems Guild</i>	
“The Role of Software Development Methodologies: Past, Present, and Future”	2
 Session 2A: Understanding and Analysis	
The Program Understanding Problem: Analysis and a Heuristic Approach	6
<i>S. Woods and Q. Yang</i>	
The Design of Whole-Program Analysis Tools	16
<i>D.C. Atkinson and W.G. Griswold</i>	
How to Identify Binary Relations for Domain Models	28
<i>H. Kaindl</i>	
 Session 2B: Supporting Requirements	
OPSIS: A View Mechanism for Software Processes which Supports their Evolution and Reuse	38
<i>D. Avrilionis, P-Y. Cunin, and C. Fernström</i>	
GRIDS — GRaph-Based Integrated Development of Software: Integrating Different Perspectives of Software Engineering	48
<i>A. Zamperoni</i>	
An Analytic Framework for Specifying and Analyzing Imprecise Requirements	60
<i>X.F. Liu and J. Yen</i>	
 Session 2C: Testing and Analysis	
Assertion-Oriented Automated Test Data Generation	71
<i>B. Korel and A.M. Al-Yami</i>	
A Specification-Based Adaptive Test Case Generation Strategy for Open Operating System Standards	81
<i>A. Watanabe and K. Sakamura</i>	
An Emprical Study of Static Call Graph Extractors	90
<i>G.C. Murphy, D. Notkin, and E.S-C. Lan</i>	

Session 2D: Industrial Experiences

An Operating System Development: Windows 3	101
<i>C. Anderson</i>	

Session 3A: Object Orientation in Use

Industrial Experience with Design Patterns	103
<i>K. Beck, J.O. Coplien, R. Crocker, L. Dominick, G. Meszaros, F. Paulisch, and J. Vlissides</i>	
Engineering an 'Open' Client/Server-Platform for a Distributed Austrian Alpine Road-Pricing System in 240 Days — Case Study and Experience Report	115
<i>S. Biffl, T. Grechenig, and S. Oberpfalzer</i>	
An Object-Oriented Implementation of B-ISDN Signalling — Part 2: Extendability Stands the Test	125
<i>A.W. van der Vekens</i>	

Session 3B: Analysis of Distributed Systems

Independent On-Line Monitoring of Evolving Systems	134
<i>N.H. Minsky</i>	
Checking Subsystem Safety Properties in Compositional Reachability Analysis	144
<i>S.C. Cheung and J. Kramer</i>	
Assertional Reasoning about Pairwise Transient Interactions in Mobile Computing	155
<i>G-C. Roman, P.J. McCann, and J.Y. Plun</i>	

Session 3C: Panel — “Why do We Care About Software Complexity?”

Panel Chair: Shari Pfleeger – *Systems/Software Inc., USA*

Panelists: Lionel Briand – *CRIM, Canada*

David Card – *Software Productivity Solutions, USA*

Norman Fenton – *City University, England*

Ross Jefferey – *University of New South Wales, Australia*

Session 3D: Mini-Tutorial

Speaker: Douglas R. Smith – *Kestrel Institute*

“Machine Support for Software Development”	167
--	-----

Session 4A: Measurement

Effort Estimation Using Analogy	170
<i>M. Shepperd, C. Schofield, and B. Kitchenham</i>	
Experiences of Software Quality Management Using Metrics through the Life-Cycle	179
<i>H. Ogasawara, A. Yamada, and M. Kojo</i>	
Analytical and Empirical Evaluation of Software Reuse Metrics	189
<i>P. Devanbu, S. Karstu, W. Melo, and W. Thomas</i>	

Session 4B: Component-Based Software

- A Case Study in Applying a Systematic Method for COTS Selection 201
J. Kontio
- System Acquisition Based on Software Product Assessment 210
J. Mayrand and F. Coallier
- Experience Assessing an Architectural Approach to Large-Scale Systematic Reuse 220
K.J. Sullivan and J.C. Knight

Session 4C: Panel — “Is the ‘Engineering’ Paradigm for Software Development Still Adequate?”

Panel Chair: Albrecht Blaser – *IWZ, University of Heidelberg, Germany*

Panelists: Heinrich C. Mayr – *University of Klagenfurt, Austria*
Günther Koch – *European Software Institute (ESI), Spain*
Günter Merbeth – *Softlab GmbH, Germany*
H. Dieter Rombach – *University of Kaiserslautern, Germany*
Klaus Tschira – *SAP AG, Germany*

Session 4D: Mini-Tutorial

Speaker: David Harel – *The Weizmann Institute of Science, Israel*
“Some Thoughts on Statecharts, 12 Years Later”

Session 5: Keynote Address

Speaker: C.A.R. Hoare – *University of Oxford, England*
“The Role of Formal Techniques: Past, Current and Future or How Did Software Get so Reliable without Proof?” 233

Session 6A: Formal Design

- Using KIDS as a Tool Support for VDM 236
Y. Ledru
- Executable Object Modeling with Statecharts 246
D. Harel and E. Gery
- Forcing Behavioral Subtyping through Specification Inheritance 258
K.K. Dhara and G.T. Leavens
- Beyond Structured Programming 268
S. Pan and R.G. Dromey

Session 6B: Configuration Management and Reuse

- Supporting the Construction and Evolution of Component Repositories 279
S. Henninger
- A New Approach to Consistency Control in Software Engineering 289
G. Heidenreich, M. Minas, and D. Kips
- Configuration Management with Logical Structures 298
Y-J. Lin and S.P. Reiss

A Generic, Peer-to-Peer Repository for Distributed Configuration Management <i>A. van der Hoek, D. Heimbigner, and A.L. Wolf</i>	308
Session 6C: Workshop Presentations	
<i>Presentations from Pre- and Post-Conference Workshops</i>	
Session 6D: Industrial Experiences	
A Standard Software Application Development: SAP R/3 <i>H. Plattner</i>	320
Session 7: Keynote Address	
<i>IEEE Computer Society and Software Engineering Institute Software Process Achievement Award</i>	
Session 8A: Process Effectiveness	
A Systematic Survey of CMM Experience and Results <i>J.D. Herbsleb and D.R. Goldenson</i>	323
DYNAMITE: Dynamic Task Nets for Software Process Management <i>P. Heimann, G. Joeris, C-A. Krapp, and B. Westfechtel</i>	331
Designing and Implementing COO: Design Process, Architectural Style, Lessons Learned <i>C. Godart, G. Canals, F. Charoy, P. Molli, and H. Skaf</i>	342
An Evaluation of Software Test Environment Architectures <i>N.S. Eickelmann and D.J. Richardson</i>	353
Session 8B: System Validation	
Scene: Using Scenario Diagrams and Active Text for Illustrating Object-Oriented Programs <i>K. Koskimies and H. Mössenböck</i>	366
System Dynamics Modeling of an Inspection-Based Process <i>R.J. Madachy</i>	376
Monitoring Compliance of a Software System with Its High-Level Design Models <i>M. Sefika, A. Sane, and R.H. Campbell</i>	387
Session 8C: Environments	
Simplifying Data Integration: The Design of the Desert Software Development Environment <i>S.P. Reiss</i>	398
Requirements for a Layered Software Architecture Supporting Cooperative Multi-User Interaction <i>F. De Paoli and A. Sosio</i>	408
Linguistic Support for the Evolutionary Design of Software Architectures <i>T.C.N. Graham and T. Urnes</i>	418

Cooperating Evolving Components — A Rigorous Approach to Evolving Large Software Systems	428
<i>R.M. Greenwood, B.C. Warboys, and J. Sa</i>	
Session 8D: Mini-Tutorial	
Speakers: Simon Gibbs and Christian Breiteneder – <i>GMD, Germany</i>	
“Large, Multimedia Programming — Concepts and Challenges”	439
Session 9: Keynote Address —	
Speaker: Victor R. Basili – <i>University of Maryland</i>	
“The Role of Experimentation: Past, Current, and Future”	442
Session 10A: Maintenance and Evolution	
Multilanguage Interoperability in Distributed Systems	451
<i>M.J. Maybee, D.M. Heimbigner, and L.J. Osterweil</i>	
Understanding and Predicting the Process of Software Maintenance Releases	464
<i>V. Basili, L. Briand, S. Condon, Y-M. Kim, W.L. Melo, and J.D. Valett</i>	
A Scalable, Automated Process for Year 2000 System Correction	475
<i>J.M. Hart and A. Pizzarello</i>	
Session 10B: Testing Algorithms	
Reducing and Estimating the Cost of Test Coverage Criteria	486
<i>M. Marré and A. Bertolino</i>	
Slicing Object-Oriented Software	495
<i>L. Larsen and M.J. Harrold</i>	
A Reliability Model Combining Representative and Directed Testing	506
<i>B. Mitchell and S.J. Zeil</i>	
Session 10C: Workshop Presentations	
<i>Presentations from Pre- and Post-Conference Workshops</i>	
Session 10D: Mini-Tutorial	
Speaker: Gerhard Fischer – <i>University of Colorado, Boulder</i>	
“Domain-Oriented Design Environments”	517
Session 11A: System Generation	
Prototypes as Assets, not Toys: Why and How to Extract Knowledge from Prototypes	522
<i>K. Schneider</i>	
User Interface Prototyping — Concepts, Tools, and Experience	532
<i>D. Bäumer, W.R. Bischofberger, H. Lichter, and H. Züllighoven</i>	
A Software Engineering Experiment in Software Component Generation	542
<i>R.B. Kieburtz, L. McKinney, J.M. Bell, J. Hook, A. Kotov, J. Lewis, D.P. Oliva, T. Sheard, I. Smith, and L. Walton</i>	

Session 11B: Dataflow Testing

A Flexible Architecture for Building Data Flow Analyzers	554
<i>M.B. Dwyer and L.A. Clarke</i>	
An Exact Array Reference Analysis for Data Flow Testing	565
<i>I. Forgács</i>	
A Demand-Driven Analyzer for Data Flow Testing at the Integration Level	575
<i>E. Duesterwald, R. Gupta, and M.L. Soffa</i>	

Session 11C: Panel — “Software: If it is so Bad, Why Does it Sell so Well?”

Panel Chair: Wladyslaw M. Turski – *Warsaw University, Poland*

Panelists: Manfred Broy – *Technische Universität München, Germany*

Tom DeMarco – *The Atlantic Systems Guild, USA*

Tony Hoare – *University of Oxford, England*

Lee Osterweil – *University of Massachusetts, USA*

David Parnas – *McMaster University, Canada*

Session 11D: Industrial Experiences

A Telecommunication Development: Siemens' Digital Switching System, EWSD	587
<i>H-E. Binder</i>	

Author Index	589
---------------------------	-----

A Scalable, Automated Process for Year 2000 System Correction

Johnson M. Hart and Antonio Pizzarello
Peritus Software Services, Inc.
304 Concord Road
Ellerica, MA 01821-3485

Abstract

As the 21st century approaches, many computer programs will begin to fail. Applications that rely on dates of any kind may simply stop working or produce incorrect results. The year 2000 problem is a matter of business importance, not just software maintenance. Program failures arise from representing calendar dates (year, month, and day) in just 6 digits, a format that allows only 2 digits for the year. The Year 2000 problem is pervasive. It occurs in calculations, comparisons and other logic involving date-related processing including date-oriented sorting and date-indexed tables. The problem occurs in data bases and files as well as in code. This paper discusses the design of an automated software tool for code and data Y2000 correction and shows how to use the tool in a large scale system correction process. The solution is discussed in the context of large, COBOL-based commercial systems. Nonetheless, our approach is fully generalizable to other languages and classes of applications.

Keywords: Year 2000, logical code analysis, code correction, code enhancement, attribute grammars, test data generation.

I. Year Representation

Our first decision in attacking the Y2000 problem [2] involved the data representation for years. There are any number of alternatives, such as adding a century indicator to a 2-digit year and representing a date as a number of elapsed days since some starting time. Based on input from our potential customers and our assessment of common practice (at least in the US), we decided to use a 4-digit year representation. Our approach will work for other date representations, but

our discussion is centered around a 4-digit year representation.

Date Representation

Existing code shows a variety of approaches to year representation. The general approaches are:

1. Ignore the century. Dates have 2-digit year fields.
2. Use a one-digit century indicator ("CI") in conjunction with the 2-digit year field. The CI might have a value of 0 for 1800, 1 for 1900, and so on. Actual usage is not consistent.
3. Use a 2-digit century indicator. Depending on the base value and field position, this approach can be equivalent to the 4-digit representation.

Advantages of the 8-digit Date with 4-Digit Year

The decision to use a true 8-digit date (four digits for the year and two each for the month and day) representation has important advantages over various shortened forms or century indicators. Compared to current practice, we found that a four digit year has advantages including:

- Compatible, consistent, and portable code.
- Readable, maintainable, and reliable systems.
- Smaller and faster code.
- Simplicity and completeness.

The only possible disadvantage of the 8-digit date is that data fields are longer (8 bytes rather than 6 or 7). This requires slightly more file space, which is not considered to be a significant enough to outweigh the benefits. If necessary, however, 8-byte date fields can be easily packed into 4 bytes before file storage.

Correction Overview

The Peritus AutoEnhancer/2000™ is built on the Peritus Code Analyzer, a rules-based tool for logical

code analysis [6, 7]. AutoEnhancer/2000 will convert existing code so that all dates containing year and/or century information are represented by a 4-digit year. AutoEnhancer/2000 will identify all appropriate date variables, change their definitions, and convert all code and constants that involve dates and date computations.

The converted code will be able to handle dates that span centuries (1800s, 1900s, and 2000s at least). Date operations will include comparisons (is one date earlier than another?), subtractions (compute age by subtracting date of birth from issue date; compute time since issue date), and additions (add surrender period to issue date). In order to meet these and other requirements, AutoEnhancer/2000 will represent complete dates with definitions such as:

```
01 ISSUE-DATE PIC 9(8).
01 ISSUE-DATE-YMD REDEFINES ISSUE-DATE.
   02 ISSUE-YYYY PIC 9999.
   02 ISSUE-MM    PIC 99.
   02 ISSUE-DD    PIC 99.
```

Years are represented by four digits, with two digits each for the century and year within the century. Also, dates are arranged from most significant (century and year) to least significant (day). Date constants will be the "true value" with no base value. For instance,

```
MOVE 19440212 TO ISSUE-DATE
```

refers to February 12, 1944, and

```
IF ISSUE-DATE GREATER THAN 20010121
```

refers to January 21, 2001.

II. Identification

The core problem in solving the Y2000 problem is to identify those variables and constants that are date sensitive so that the code and data files can be corrected. The AutoEnhancer/2000 approach is an extension of the type checking that is done by compilers for many programming languages. In conventional programming languages, however, types are only checked for values such as integer or real, or aggregates (arrays, structures) of these types. Type conflicts are normally resolved by converting data representations or by generating an error. AutoEnhancer/2000 Y2000 type checking and propagation are more extensive.

Syntax-directed translation and type checking are explained in Chapters 5 and 6 of *Compilers: Principles, Techniques, and Tools* [1]. The underlying concept is the *attribute grammar*.

A *syntax-directed definition* or *attribute grammar* is a generalization of a context-free grammar (or BNF) used to define the programming language. From [1], p. 280:

Each grammar symbol has an associated set of attributes, partitioned into two subsets called the synthesized and inherited attributes of that grammar symbol. If we think of a node for the grammar symbol in a parse tree as a record with fields for holding information, then an attribute corresponds to the name of a field... The value of an attribute at a parse-tree node is defined by a semantic rule associated with the production used at that node. The value of a synthesized attribute at a node is computed from the values of attributes at the children of that node in the parse tree; the value of an inherited attribute is computed from the values of attributes at the siblings and parent of that node.

In general, the synthesized (or *bottom-up*) attributes are the easiest to compute, although the *top-down* inherited attributes are sometimes necessary. For example:

```
MOVE BILLING-DATE-YR TO DURATION-WS
```

causes DURATION-WS to receive a synthesized attribute from BILLING-DATE-YR, or it causes BILLING-DATE-YR to receive an inherited attribute from DURATION-WS. How this attribute propagation is done is determined by what attributes are already known. In

```
IF YR-OF-INCREASE > 84
  COMPUTE YR-OF-INCRSE-PLUS1 =
    YR-OF-INCRSE-YR + 1
```

the two constants receive inherited attributes from the variables they are combined with and assigned to.

Definition of Terms

1. *variable* refers to named variables and structures, regardless of whether they are I/O fields, working storage, or occur in the linkage section.
2. *I/O variables* are fields defined as part of a file record, a CICS screen, or JCL.
3. A variable is *date-sensitive* if its format attribute contains *y* (year) or *c* (century).
4. An *initial date seed* is a date I/O variable. An initial seed must be an actual *date* and not an elapsed period of time, such as an age or policy term.
5. A *derived date seed* is a date-sensitive I/O date variable found during the AutoEnhancer/2000 Identification phase.

The Identification Method

Date-sensitive variables and constants, then, are those that require new declarations, modified code logic, or other changes in their use when converting a program from 2-digit to 4-digit year representations. Most

proposed solutions to the Y2000 problem are based on using pattern matching to create a "suspect list" of variables that appear to be date-sensitive. Pattern matching tools can be put together very quickly using UNIX utilities such as `grep`, but pattern matching has a number of shortcomings:

- *False positives.* The variable `PREMIUM-YEAR-TO-DATE` would be incorrectly added to the suspect list as "YEAR" and "DATE" are obvious patterns for a date-sensitive variable.
- *False negatives.* The variables `TERM` and `DURATION-WS` below are examples, where we have statements such as:

```
MOVE BILLING-DATE-YR TO DURATION-WS
SUBTRACT ISSUE-YEAR FROM DURATION-WS
COMPUTE TERM = TERM - WS-OPTION-YEARS
```

- Date-sensitive *constants* are ignored. Consider this code, where 84 probably means 1984, 30 may or may not be 1930, and 1 almost certainly refers to an elapsed time of one year:

```
MOVE 30 TO TERM
IF YR-OF-INCREASE > 84
  COMPUTE YR-OF-INCRSE-PLUS1 =
    YR-OF-INCRSE-YR + 1
```

- Date-sensitive variables and constants are not identified according to their context. A constant should be identified as date sensitive by virtue of the statement it occurs in and the variables that it is combined with rather than by its name alone.
- A pattern-matching suspect generation strategy only identifies the names that are suspected of being date-sensitive. It is still necessary to determine manually which suspects are date-sensitive and to correct their definitions, and it is also necessary to correct some statements that they occur in, including I/O statements.

Attributes as Used by AutoEnhancer/2000

These are the specifications for the attributes that are maintained with each symbol and constant in the AutoEnhancer/2000 Identifier symbol table (global and local attribute database) during the *front end* and *identification* stages. Each attribute and its value range is specified, along with basic consistency rules.

1. Format attribute

Values: century or c
year or Y
month or M
day or D
zero or z for unknown

- a) Combinations of these, with each attribute letter indicating a byte, such as:

```
YYMMDD
CYY
MDD
```

- b) The order is important, and this scheme allows a date representation such as:

```
MMDDYY
```

which might be generated for a report, whereas the data may be stored internally in the more common `YYMMDD` format.

- c) Structure elements are represented by sequence of format/offset pairs, where the offset is the byte position within the structure. A structure with four fields, in order:

```
CYY
YYMMDD
DD
```

is represented by the format/offset pairs:

```
<CYY, 0>
<YYMMDD, 3>
```

```
<DD, 10>
```

2. A **date** is distinguished from a date-sensitive **duration** (an elapsed period of time). Identification rules specify, for example, that the subtraction of two **dates** produces a **duration**, or adding a **date** and a **duration** produces a **date**.

3. Base attribute

Values: Integers corresponding to the *Format*, such as: 1800 for a `CYY` attribute.

AutoEnhancer/2000 maintains these attributes in a variable length string associated with each variable.

More Theory and Some Practice: Locally and Globally Equivalent Variables

During initial lexical and syntactical code analysis in the AutoEnhancer/2000 Front End, AutoEnhancer/2000 creates a database of variable relationships called the L, or "local equivalent" relations. Two variables are locally equivalent if they must have the same format and base. Only these attributes are equivalent; the variables may be totally distinct.

1. Variables X and Y are related by L (written as L[X, Y]) if they must have the same format attributes.
2. AutoEnhancer/2000 enters the fact that L[X, Y] whenever X and Y occur and "interact" through `REDEFINES`, I/O, or in the same expression, such as:

```
MOVE X TO Y
COMPUTE X = Y + 2
```

3. L is an "equivalence relation." It does not matter if you write $L[X, Y]$ or $L[Y, X]$. Furthermore, L creates chains of equivalent formats, so that if $L[X, Y]$ and $L[Y, Z]$, then $L[X, Z]$. This chaining is the key factor driving the AutoEnhancer/2000 Identifier.

There is a similar "global equivalent" relation, G. Whenever two variables match through linkage sections, AutoEnhancer/2000 says they are globally equivalent. This determination, however, occurs later.

The goal is to find all variables that are globally (G) and locally (L) equivalent. We then compute the *transitive closure* of the G and L relationships to find the entire set of variables that have equivalent format. In mathematical terms, we created a relation called F to denote equivalent formats. We compute the closure of the L relation, add the result to the current G relation, take the closure of the result and then repeat the entire local and global closure cycles to get F. This is:

$$F = (L * \text{union } G)^* \text{ which is the same as } (L \text{ union } G)^*$$

This simply means that we trace all the equivalent format relationships through the code and keep on tracing until every date-sensitive variable is accounted for. The "union" is set union and the * is the "transitive closure" operator which traces these relationships. "Transitive" means that we trace the relationships, and "closure" refers to the fact that we trace the transitive relationships until there are no changes, or the relationship is "closed."

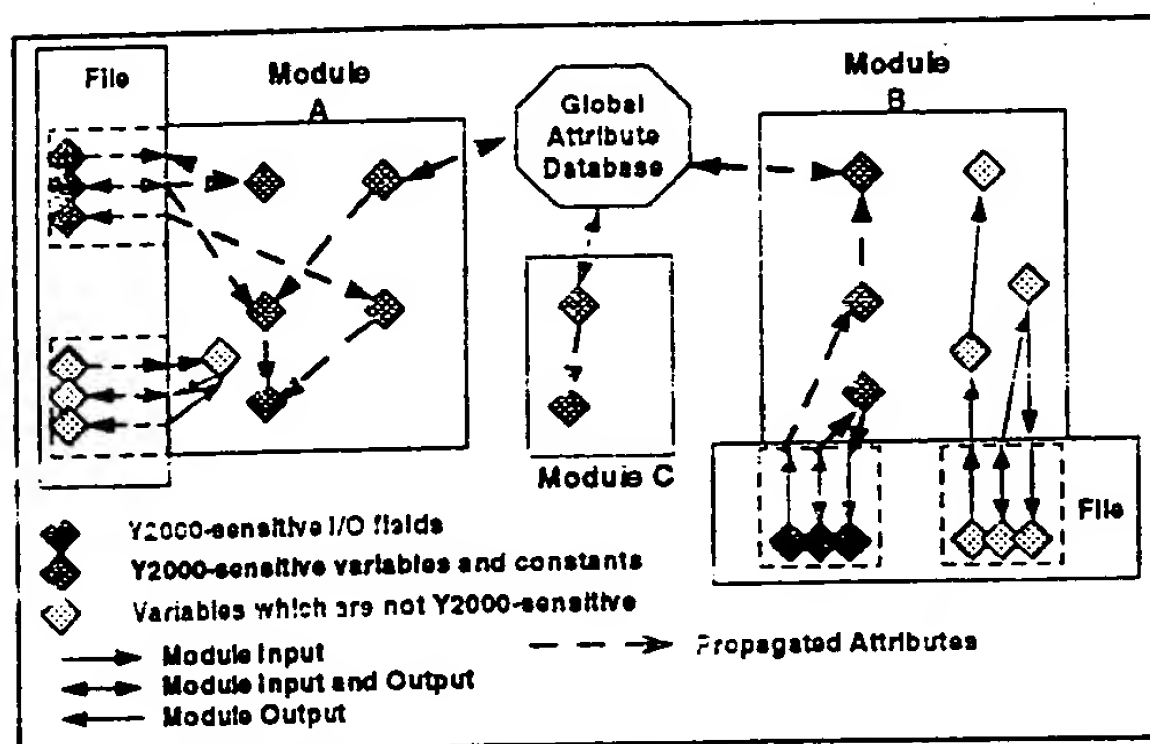


Figure 1. The AutoEnhancer/2000 Identification Phase Using Initial Seeds and Transitive Closure

We also perform "system closure" which passes format information from one program to another using the formats of shared files. System closure requires processing of JCL (Job Control Language) to determine the files that are accessed by the programs.

Putting the Theory Into Practice: Attributes for Y2000 Identification

In practice, the AutoEnhancer/2000 Identifier does not actually compute F. It propagates the formats to equivalent variables, which is the desired result. At the end, all equivalent variables have the same format.

AutoEnhancer/2000 has three stages, the Front End (FE), the Identifier, and the Corrector.

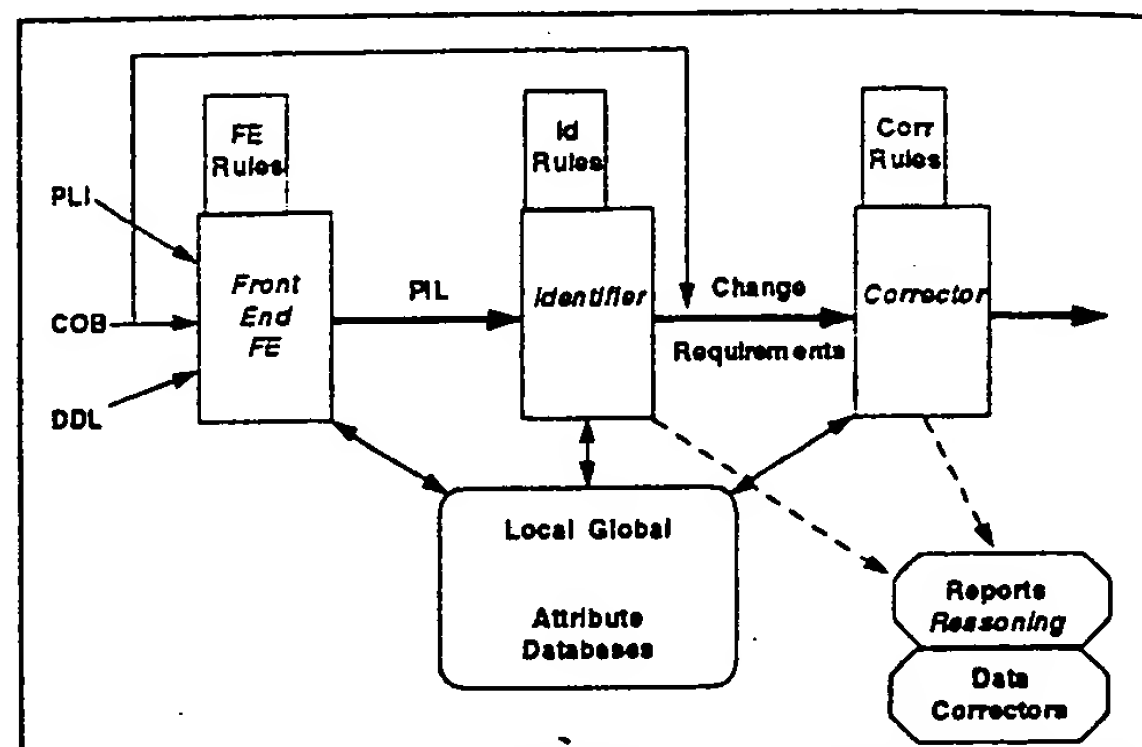


Figure 2. The AutoEnhancer/2000 Front End, Identification and Correction Phases

During the AutoEnhancer/2000 Front End (FE), the L relationship is initialized. The PIL (intermediate language) is generated to be used by the Identifier.

The AutoEnhancer/2000 Identifier adds more L and G relationships (what it really does is make the formats the same) by examining the code of a compile module:

1. The *initial seeds* establish the date formats for some I/O variables. Initial seeds are specified by the user based upon well-known information, such as the dates that appear on a screen or report. Once common shared files are formatted, very little additional seeding is required.
2. Every time variables interact through subroutine calls or shared files, they are added to G.
3. Every occurrence of a **REDEFINES** creates an L relationship between the variables and fields within structures.
4. Every move of a part of a record or array creates an L relation between the target and destination fields.
5. Whenever a variable is truncated so that the YY part of a variable is lost, any existing L relation between the variables is removed.
6. At every step where an addition is made to G or L, the Identifier updates the transitive closure, F, and propagates attributes among related variables. A report "reasoning" entry is also generated. It is possible to trace the format of any variable.

The AutoEnhancer/2000 Identifier continues this process through all the compile modules until there are

no changes to any formats. This may require several passes through the compile modules to account for all the subroutine linkages.

When the process is complete, the attribute databases contain all the information on variables of all types. From this, the derived seeds (both date and non-date) can be determined directly from the format attributes. Format attributes are also used as input to the correction phase.

The AutoEnhancer/2000 front end uses standard compiler techniques [1, 8, 9]. The rules-based subsystem is implemented using Clips [11]. The intermediate language (PIL) is based on Dijkstra guarded commands [3, 4, 5].

III. Code Correction

The AutoEnhancer/2000 Code Correction Phase follows the Identification Phase. Code Correction uses correction requests generated during Identification. Each correction is driven by a "correction rule" which is invoked when certain code usage patterns are identified.

• Indicates	Input Seeds (I/O Variables)		
• DATE-A	Y2000-sensitive	Format: YY	Date
• TERM-B	Y2000-sensitive	Format: YY	Duration
Original Code:			
MOVE 86 TO DATE-A			
IF DATE-A < 90 COMPUTE DATE-A = DATE-A + TERM-B			
DATE-A and TERM-B are corrected to PIC9(4) (for YYYY) in the data division			
Corrected Output COBOL code in the procedure division:			
MOVE 1986 TO DATE-A			
IF DATE-A < 1990 COMPUTE DATE-A = DATE-A + TERM-B			

Figure 3: AutoEnhancer/2000 Correction Rules In Operation I

Upon completion of Identification, the Attribute Database contains an entry, with format, for every variable. From this information, the Corrector can determine which variables are date-sensitive (containing year and/or century information) and which of those are dates (being actual dates and not elapsed times, such as an age). The correction rules used are invoked based on the actual "Old Code" and the attributes of the variables involved that occur in a line, or group of lines, of code.

Each correction rule must change the existing "Old Code" (OC) into "New Code" (NC) such that:

- The New Code properly deals with 4-digit years rather than 2-digit years and century indicators.
- The New Code preserves the logic of the Old Code.
- The New Code contains constants that properly represent 4-digit years, but the corrector must not

adjust OC constants that do not represent years and centuries.

- Any OC dealing with century indicators must be removed, but any logic dealing with century indicators must be preserved in the 4-digit year logic.

The following information is provided for each rule:

- The rule name.
- An informal, English language, description of the rule.
- One or more examples of using the rule.
- A proof that the rule is correct. That is, the NC and OC behave in exactly the same way.
- The actual rule, expressed in the AutoEnhancer/2000 rule language [10].

The rule description and examples are used to understand the nature of the code changes, conformance to coding conventions, etc. The correction rules are grouped into three categories, corresponding to the order in which the AutoEnhancer/2000 corrector applies them.

- *Isolation Rules* separate the date-sensitive and non-date-sensitive variables. On completion, the Attribute Database will contain all date-sensitive variables and their formats.
- *Data Division Correction Rules* correct the data division, primarily merging year and century fields and widening fields to accommodate 4-digit years.
- *Procedure Division Correction Rules* correct the code, primarily adjusting constants and removing century indicator logic.

Within each category, many of the rules deal with overhead tasks such as initialization or printing results, so they do not require proofs or extensive explanations.

• Indicates	Date		
• DA, DB, *DC	are Y2000-sensitive	Format: CYY	CenturyBase = 18
Original:			
MOVE 185 TO DA			
COMPUTE DB = DA - 5			
MOVE DB TO DC			
DA, DB, DC are corrected to PIC9(4) (YYYY) in the data division			
Final:			
MOVE 1985 TO DA			
COMPUTE DB = DA - 5			
MOVE DB TO DC			

Figure 4: AutoEnhancer/2000 Correction Rules In Operation II

Experimental Results: Our experimental results for identification and correction have been very favorable. For example, in one study, we performed identification and correction on a program of 300K lines of code (KLOC) in 1,170 compile modules (all numbers are approximate). We started with 90 seeds (all file record fields) and identified 600 date-sensitive file record fields

and 7,000 date-sensitive working storage (internal) variables, all of which were confirmed manually. The entire process took about five hours on UNIX workstations, which is comparable to compile time.

IV. The System Correction Process

One could assume that corrected code can only access corrected data, and corrected data can only be accessed by corrected code. This situation creates the conceptual "Big Bang" System Correction (BBSC) process.

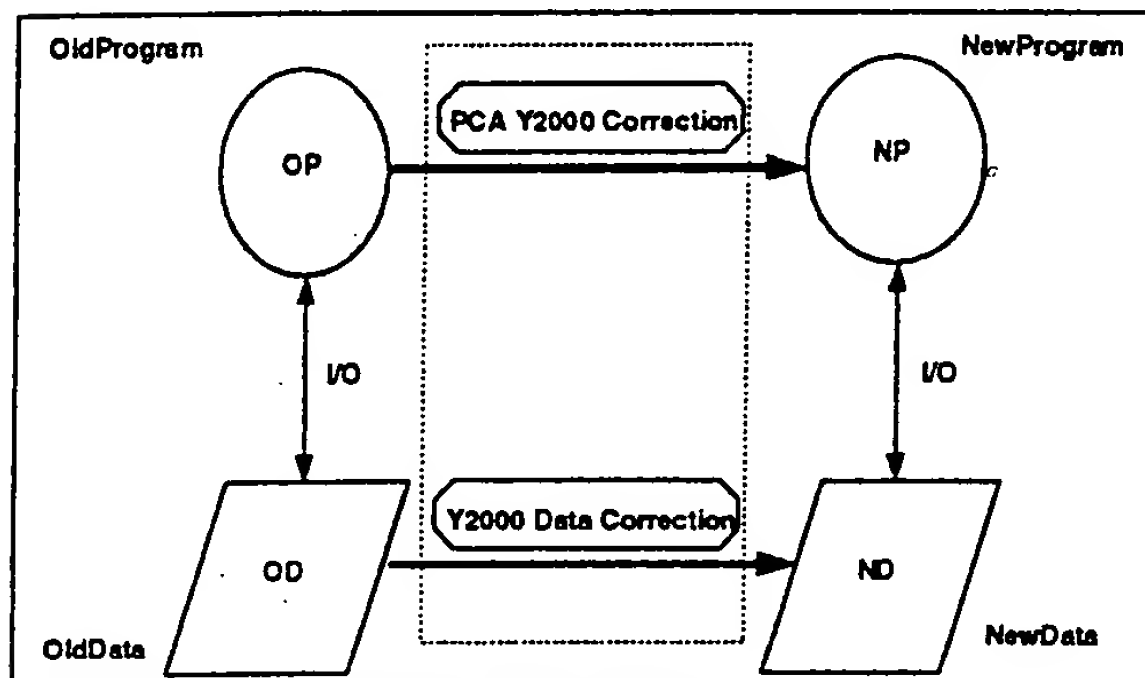


Figure 5: Big Bang System Correction

Big Bang correction is useful to illustrate system correction at a high level. However, it is not practical to carry out a BBSC, except for very small systems, for the following reasons:

- *Size and Scale.* The Old Program can consist of tens or hundreds of millions of lines of code and the Old Data will be thousands of gigabytes. Correction, even with the fastest tools, will take weeks or even months.
- *Disruption.* During correction, all operations, including code maintenance and mission critical applications, must be shut down. This is not feasible.
- *Testing and Reliability.* There is no opportunity to perform incremental unit tests on the corrected code working with corrected data, and a double set of hardware would be required to test the Old and New Systems in parallel.
- *High Risk.* BBSC has no opportunity for recovery.

The obvious alternative is to correct portions of the code step-by-step. The difficulty, however, is that some data files and databases are required by nearly all program subsystems. In order to test a corrected subsystem, you must correct all the files and databases that the subsystem accesses. Then, however, the uncorrected programs cannot operate.

AutoEnhancer/2000 Correction software consists of two principal components which must be combined:

- AutoEnhancer/2000 proper, for program correction. Program correction provides verified Y2000-correct code, test data points, and a complete attribute database for all working storage, record field, and screen field "variables."
- *Data Correctors* of two types which convert between Old and New data formats:
 1. *Batch Sequential Data Correctors* (BSDCs) which are used for pipeline and one-time correction. BSDCs can be bi-directional; there can be two separate BSDCs for a single file. The first will correct OD to ND, and the other will correct ND to OD. Of course, ND to OD correction will be erroneous if the ND file contains a 21-st century date.
 2. *Wrapper Data Correctors* (WDCs) which are subroutines used for dynamic, runtime, access, performing data correction between two and four-digit representations.

Data Correctors

Data Correctors are programs (either BSDCs or WDCs) which are generated by AutoEnhancer/2000 from format and other information in the Attribute Database.

- There will be a distinct data corrector set for each file or file family. A *file family* is a collection of files determined by AutoEnhancer/2000 to have exactly the same record formats.
- A *data corrector set* consists of up to two BSDCs (one for each direction) and a WDC. In many cases, depending on usage, AutoEnhancer/2000 will not need to create all three data corrector programs for a given file family.
- As each file family has its own distinct set of data corrector programs, it will be possible to customize the *core* correctors (generated by AutoEnhancer/2000) to account for factors not detectable by AutoEnhancer/2000. Some record fields may be used as both a date and a non-date, with the interpretation determined by the value in another field (such a situation is called a *discriminated union*).

Batch Sequential Data Correctors

Figure 6 shows the operation of a BSDC program. The solid arrow shows the OldData to NewData correction, while the dashed arrow shows the reverse NewData to

OldData correction. Figure 6 shows records from a hypothetical database where each record contains three dates, each with the format CYYMMDD, where the century indicator base is 1800.

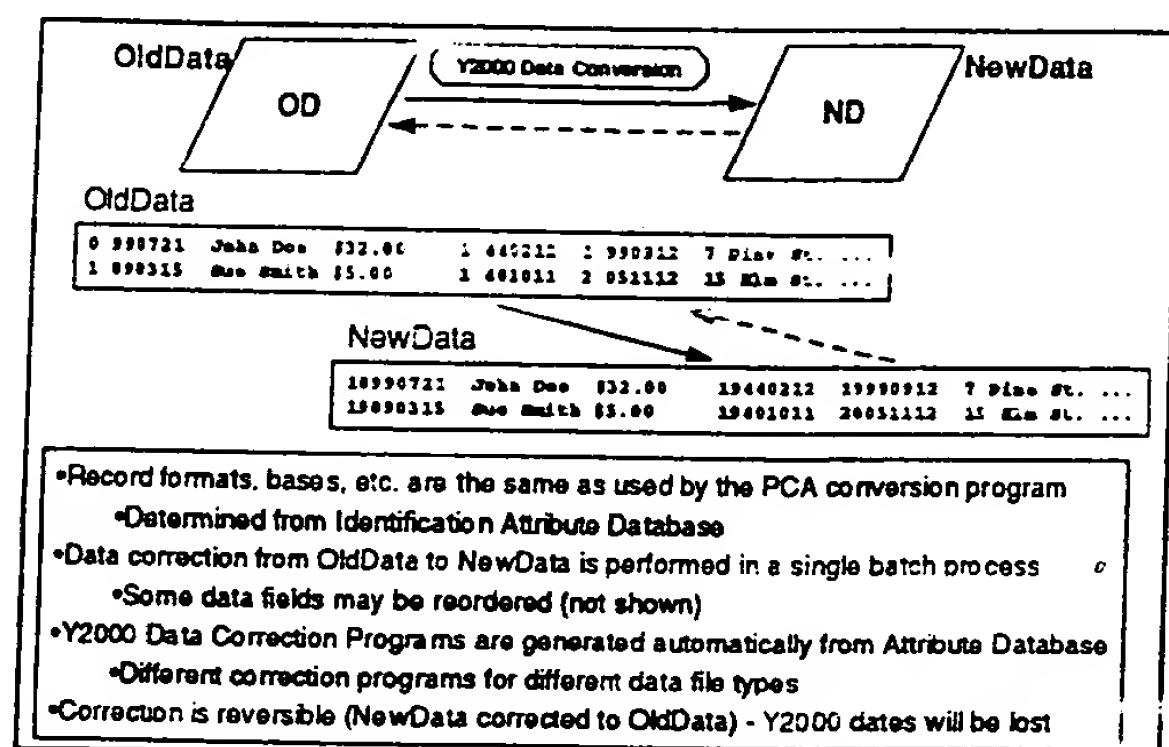


Figure 6: Batch Sequential Data Correction

BSDCs are suitable for any file that is processed sequentially or for one-time, shared file correction, such as a master account file. One-time, shared file correction is appropriate after all programs that access the file are corrected.

BSDCs are bi-directional, allowing recovery in case of program correction errors or any other problems in the system correction process.

Figure 7 is an example of a pipeline using BSDCs. Programs are shown in ovals and are marked as *New* or *Old*. New programs have been corrected (by AutoEnhancer/2000) and read data in 4-digit year format. Old programs are still in their original form. Parallelograms indicate files, in either *Old* or *New* format. The shaded, pointed boxes indicate BSDC programs which read files sequentially in one format and rewrite them in the other format.

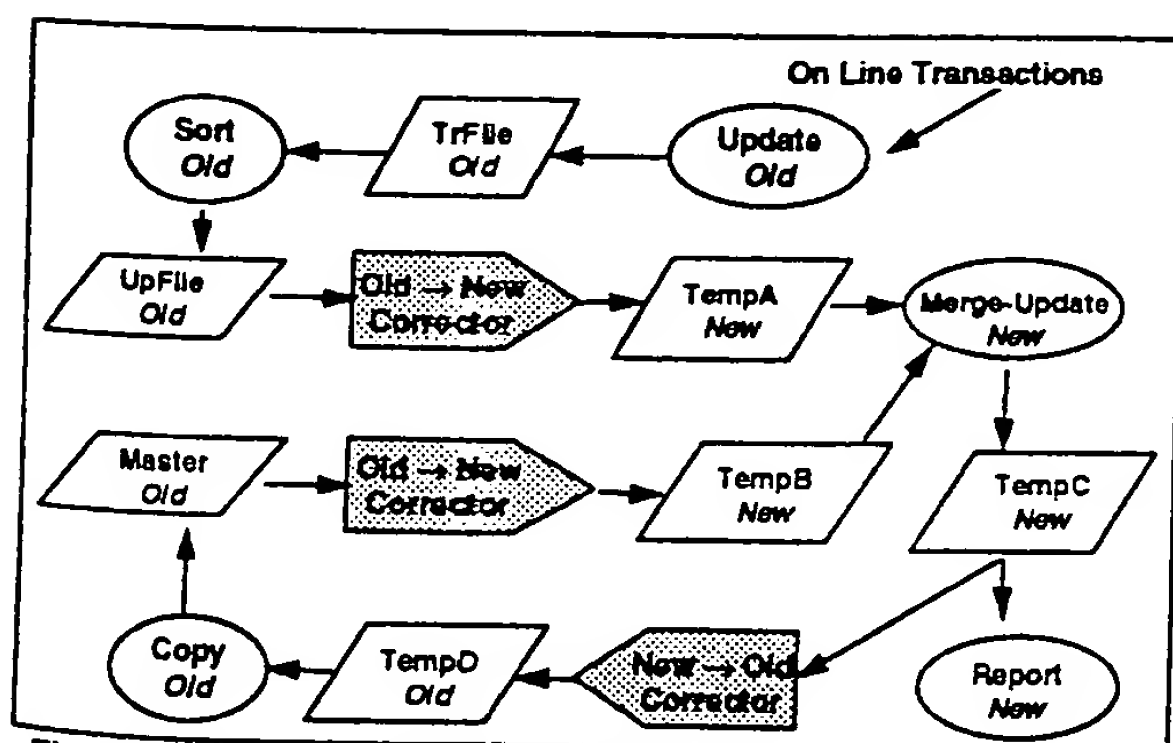


Figure 7: Data Correction Pipelines

Tracing the operation of the Figure 7 example, which performs a periodic update of a sorted master file based upon update transactions, illustrates the utility of BSDCs in a data correction pipeline. Some programs have been corrected, and others have not. The commands to execute the BSDCs must be inserted into the JCL (Job Control Language) program that executes this cycle. Notice that JCL correction is part of the overall AutoEnhancer/2000 program correction process.

Run-Time Wrapper Access

BSDC cannot meet all requirements, however. The primary limitations are:

- Files that are accessed *concurrently* by both corrected and uncorrected programs.
- Files that are accessed *randomly* rather than sequentially. For example, if a corrected program accesses an uncorrected data file, the data correction must be performed "on-the-fly" rather than as part of a batch process.

The solution is for AutoEnhancer/2000 to generate "wrapper routines" (WDCs) during program correction. AutoEnhancer/2000 replaces I/O operations with calls to the WDCs. WDCs, in turn, perform the actual I/O and correct dates (convert from 2-digit, possibly with a century indicator, to 4-digit representations) during reads and to convert dates back to 2-digit form during writes. In this way, correction is performed dynamically, and the files can be shared with uncorrected programs.

Notice that WDCs are only used by corrected programs as AutoEnhancer/2000 changes normal COBOL I/O to be calls to the WDCs, and AutoEnhancer/2000 also generates the WDC code. WDCs allow corrected programs to read and write uncorrected data files, allowing corrected and uncorrected code to *coexist* so long as no 21-st century dates are used. In this way, we achieve *modularity* and *incremental testability*.

After all accessing programs are corrected, wrappers can be removed.

Figure 8 illustrates Old and New Program code where the New Program generated by AutoEnhancer/2000 invokes the WDCs named WR-READ-YEAR and WR-WRITE-POLICY-RECORD.

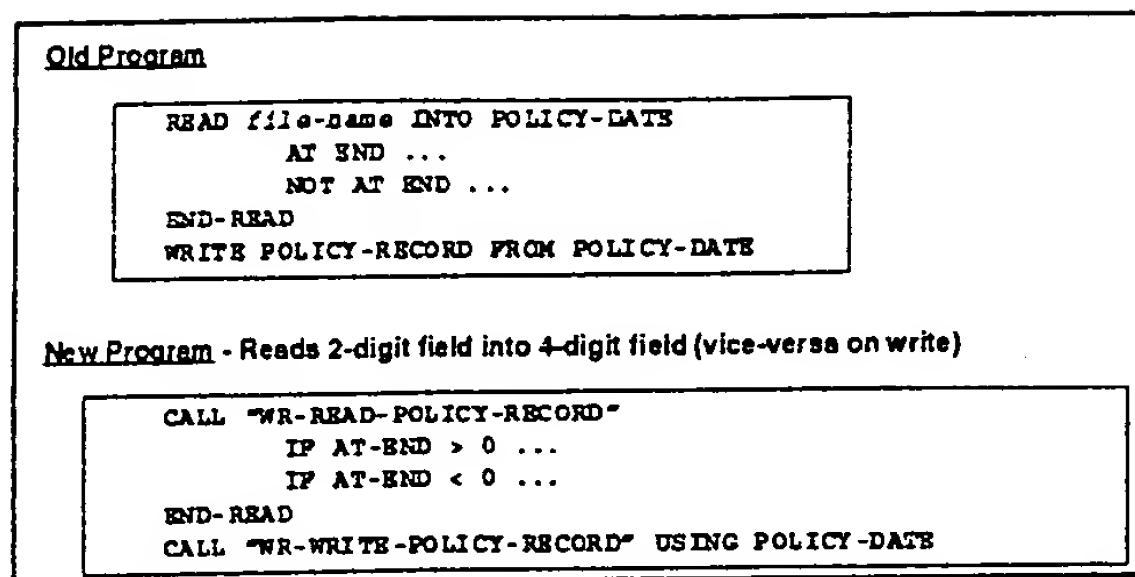


Figure 8: Run-Time Wrapper Access Code

There are a number of different WDC strategies, of which this is one. In particular, when the COBOL program uses a database management system (DBMS) for I/O, rather than making direct READ and WRITE calls, the WDCs will replace the DBMS calls. The WDCs, in turn, will invoke the DBMS.

Likewise, there are several strategies for WDC removal after all programs are corrected.

1. *Relink* with dummy WDCs that do not perform any data correction.
2. *Recorrect*, directing AutoEnhancer/2000 not to replace I/O operations.
3. *Dynamic removal*. In this scheme, each WDC contains a flag telling it whether or not to perform data correction.

The first strategy (using dummy, or "pass through" wrappers) dynamically linked, is the approach we have selected. This is the least disruptive process, and there is a long-term advantage in having all I/O be "externalized" from other code (for example, graphical user interface replacements will be much simpler).

Incremental Program Correction and Globes

With BSDC and WDC it is now possible to develop a strategy for the System Correction Process (SCP). The key is to convert a single "glob" at a time. The desirable characteristics of a glob are:

1. A glob includes a manageable unit of program code corresponding to a significant program.
2. The glob also includes all data files that are *local* to the code. A data file is local if it is accessed exclusively by code in the glob.
3. A glob is of a size that can be conveniently corrected in one step.
4. A million lines of code (1 MLOC) appears to be an appropriate size based on the need to correct the entire code base in a timely manner.
5. Each glob should be closely related to previously corrected globs and should be selected to maximize the amount of corrected data.

Figure 9 shows the first step. Glob A has been identified, and it consists of both program code and associated local data (shown as A-Data). AutoEnhancer/2000 corrects the code and generates both the WDCs (shown as W) and the BSDC "Y2000 Data Correction" program. W accesses all data that must be shared at runtime with uncorrected code.

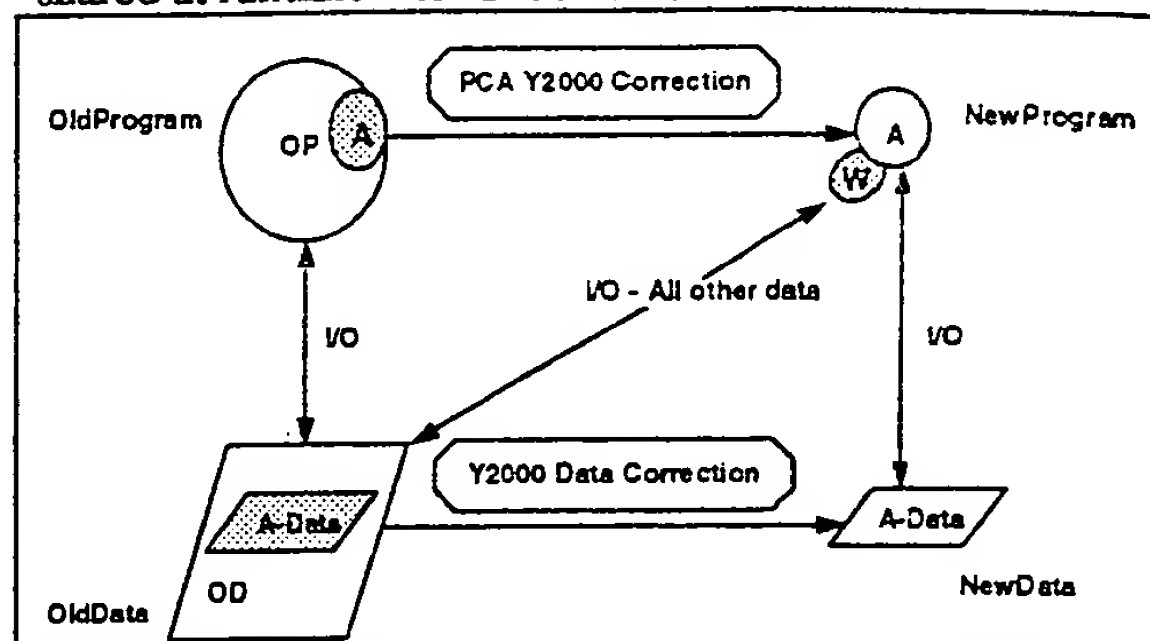


Figure 9: Run-Time Wrapper Access: "Correcting Glob A"

At this point, the system is still not operational as W only accesses shared, random access files. Therefore, AutoEnhancer/2000 next corrects JCL statements that invoke programs within the Glob A code by inserting BSDC programs for pipeline correction. Figure 10 shows the resulting operational, partially corrected, system.

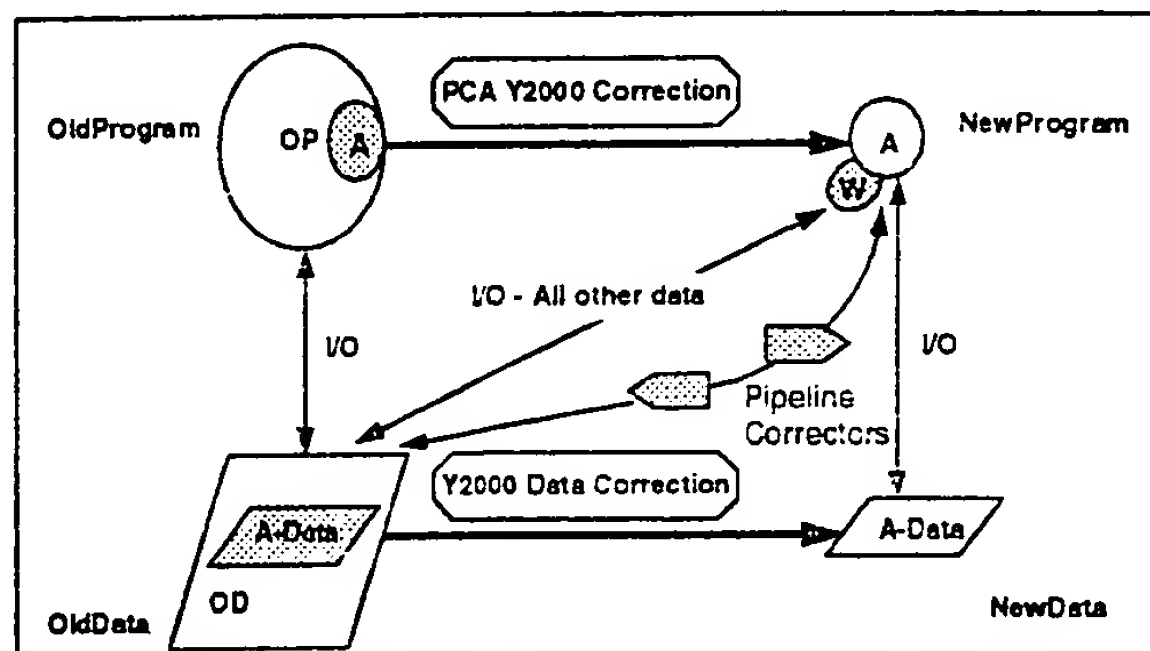


Figure 10: Run-Time Wrapper Access: "Using Pipelines"

Next, Glob B is corrected, and AutoEnhancer/2000 attaches the B code to its own set of WDCs. In addition, B-Data includes not just the data that is local to B but also that which is local to A and B together. Figure 11 shows the resulting partially corrected system.

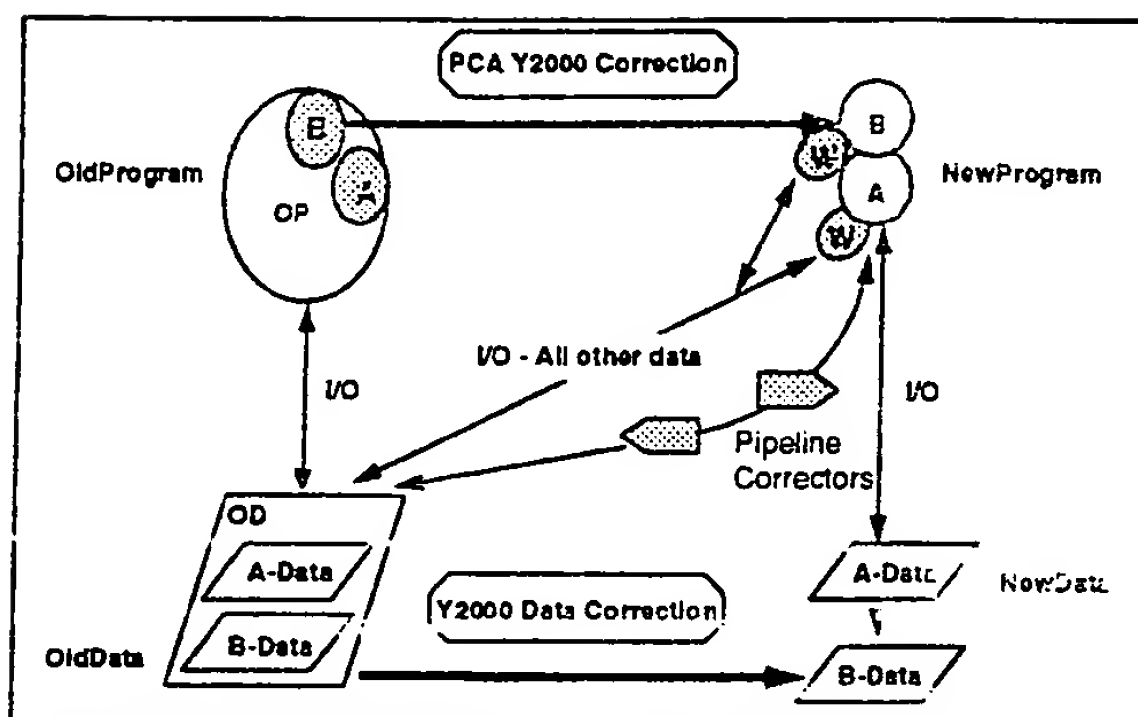


Figure 11: Run-Time Wrapper Access: "Correcting Glob B"

Glob-by-glob, the process continues until all code is corrected, as shown in Figures 12 and 13. At this point, WDCs are no longer needed, so they are removed. All code and data are corrected.

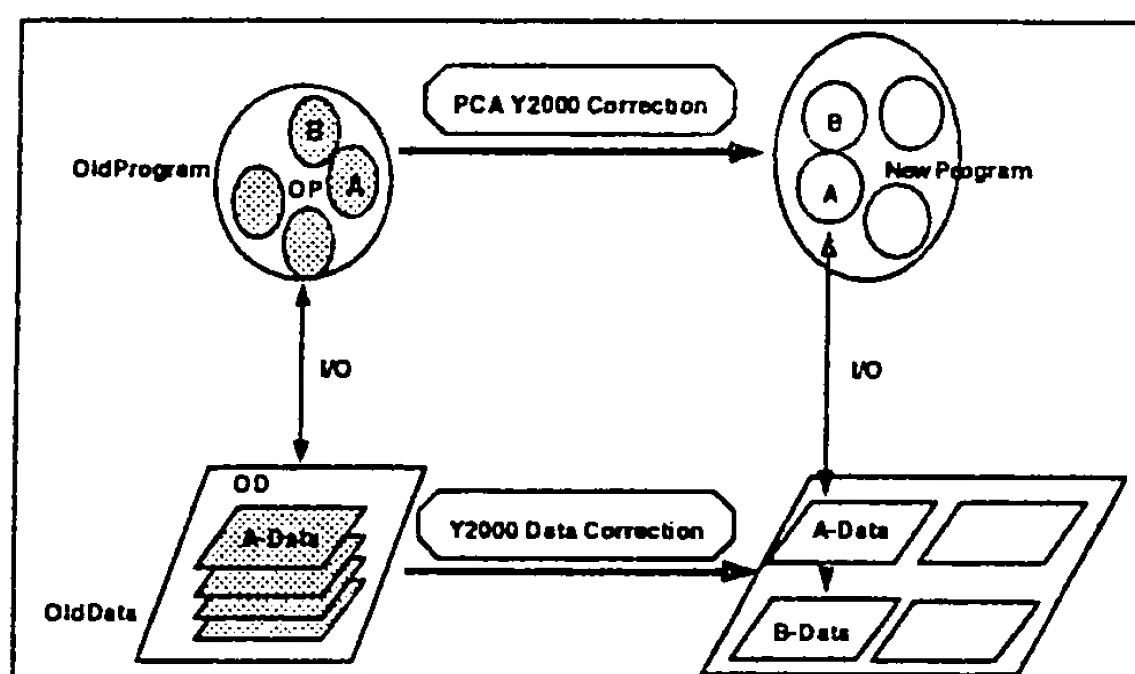


Figure 12: System Migration: "Correcting Glob Z"

Figure 13 now shows the final, fully corrected system. Archived data is not corrected until it is required, and, even then, it is only corrected into temporary read-only files for use by the New Program. The archive files themselves are not changed.

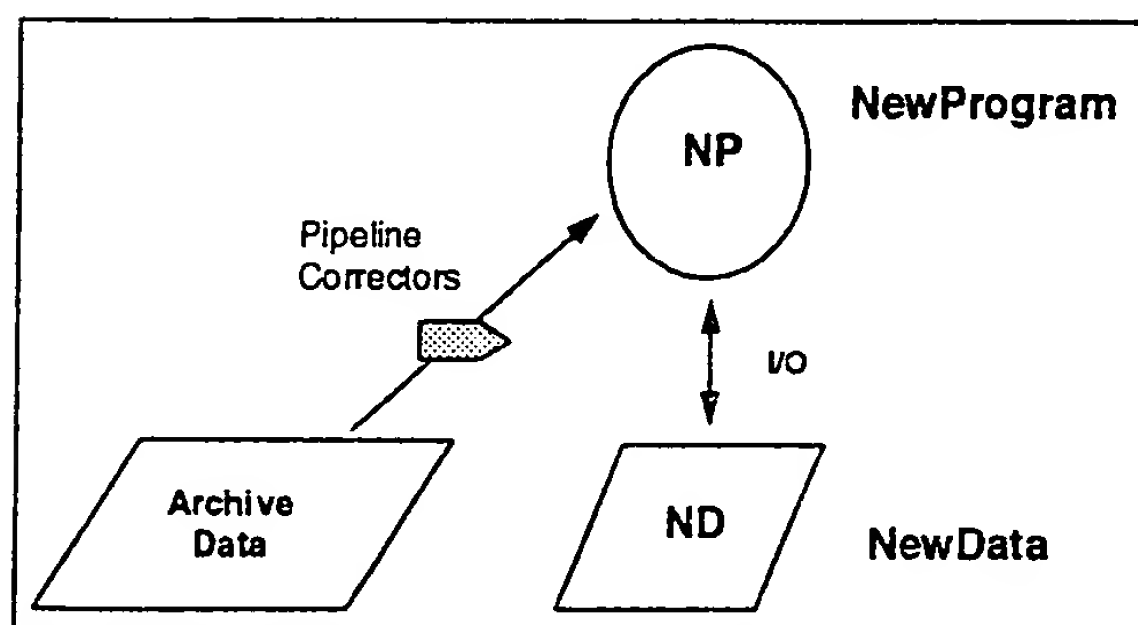


Figure 13: System Migration - Completed

Features and Advantages of Incremental Correction

1. *Testing.* Each corrected glob can be tested. 21-st century test data can be introduced early.
2. *Production.* Corrected programs and data can be put into production after passing initial tests and will be exposed to considerable real-life testing. Any early defects will benefit future corrections.
3. *Reliability and Recovery.* This process allows for back-up of any step and recovery.
4. Data is corrected as soon as possible.
5. BSDCs, and if necessary, WDCs, can access archive data.

There are several tasks to be performed in order to implement this SCP.

1. The various forms of WDC need to be specified, including the mechanisms for removing wrappers.
2. We must specify and implement a policy for identifying globs and determining the order of correction. Factors to consider include:
 - *Urgency*
 - *Complexity and Risk.* The most complex and risky programs should be corrected first
 - *Connectivity.* Programs which are most tightly connected (through data files) are candidates for the next correction step.
3. AutoEnhancer/2000 generates the data correction programs. In addition, AutoEnhancer/2000 determines which files are appropriate for BSDCs and which require WDCs.
4. Analysis of CICS (user interface) and JCL ("Job Control Language") is necessary.

Verification

The correctness of the transformation is shown in two steps.

1. *Rule Correctness:* The correction rules developed for the Correction phase are shown to be logically correct. That is, the set of individual rules transform "Old Code" (OC) into "New Code" (NC), and the OC and NC have exactly the same logic and behavior, except that the NC deals with 4-digit years.
2. *Correct Rule Application:* The Correction rules are actually applied by AutoEnhancer/2000 correctly in each instance so that the underlying logic is preserved.

Both Rule Correctness and Correct Rule Application are established using AutoEnhancer/2000's techniques of *Logical Code Analysis* employing *weakest precondition* (wp) calculations [3, 4, 5, 6, 7].

Verification only assures that the OC and NC have the same logic. Neither AutoEnhancer/2000 Correction nor

Verification will correct defects in the OC. Rather, AutoEnhancer/2000 Correction assumes that the OC is correct (within the limitations of its ability to represent dates) and extends the OC to allow use of 4-digit years.

Testing

It is necessary to test the integrated New Program/Data operation. Verification correctness assumes that all date sensitive variables have been correctly identified. No date-sensitive variables can be missed, and, conversely, all variables so identified must really be date-sensitive. It is also necessary to observe system behavior for dates that exceed 2000. In order to assist testing, AutoEnhancer/2000 generates test data specifically for each procedural code change made during correction.

The objective is to generate test data that will exercise the code changes made during code correction. The test data is required both to test that the code logic has not changed (*regression testing*) and that the code logic operates correctly for dates that exceed Y2000 (*extension testing*).

Test data generation is an integral part of the AutoEnhancer/2000 Code Correction Phase. Figure 14 shows test data generation and the use of the test data.

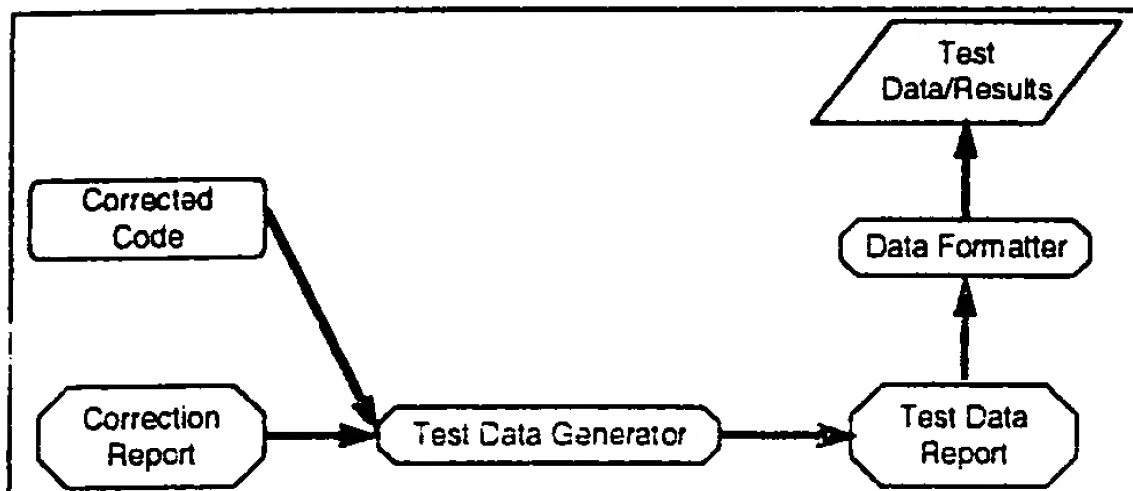


Figure 14: Report and Test Data Generation During Code Correction

1. The Corrected Code and Correction Report serve as input to the Test Data Generator. This is all done internally to AutoEnhancer/2000 operation.
2. The result is the Test Data Report, which gives the actual test data point values along with the reasoning that led to the generation of that data.
3. The Data Formatter is a system to process the Test Data Report and to insert the test data points into valid transactions and database records.
4. The Test Data/Results represent the output of the Data Formatter.

V. Summary

Using AutoEnhancer/2000 for identification, code correction, data migration, and test generation automates

the entire Year 2000 correction process, resulting in greater efficiency and higher quality than any other alternatives. Our approach overcomes limitations of other tools and techniques and:

- Addresses the entire problem
- Minimizes risks
- Increases quality

We are currently applying this system for several clients, and we are continuously refining the process based on experience.

Note: There is a U.S. patent pending on the material in this document.

Acknowledgments: Ashraf Afifi and Dominic Chan have been active contributors to the ideas in this paper, and the former is the primary implementor. Elissa Armour developed the draft of this paper from a mass of technical memos, diagrams, and notes.

References:

1. Aho, Sethi & Ullman, *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.
2. R.S. Arnold: *Resolving Year 2000 Problems in Legacy Software*. Presentation at the Eighth International Software Quality Week, San Francisco May 30 - June 2, 1995.
3. E. Cohen, *Programming in the 1990s*. NY: Springer-Verlag, 1990.
4. E. W. Dijkstra, *A Discipline of Programming*. Englewood Cliffs, NJ: Prentice-Hall, 1976.
5. D. Gries, *The Science of Programming*. NY: Springer-Verlag, 1981.
6. J. M. Hart, "Experience with logical code analysis in software reuse and re-engineering," in *AIAA Computing in Aerospace 10* (San Antonio, TX), Mar. 28-30, 1995, pp. 549-558.
7. J. M. Hart, "Experience with logical code analysis in software maintenance," *Software Practice and Experience*, vol. 25, no. 11, 1243-1262, 1995.
8. J.R. Levine, T. Mason, D. Brown: *FLEX & YACC*. O'Reilly and Associates, 1992.
9. *BISON*, Free Software Foundation, 1993.
10. *Clips 6.0 C-Language Integrated Production System*. L.B. Johnson Space Center Software Technology Branch, Undated.

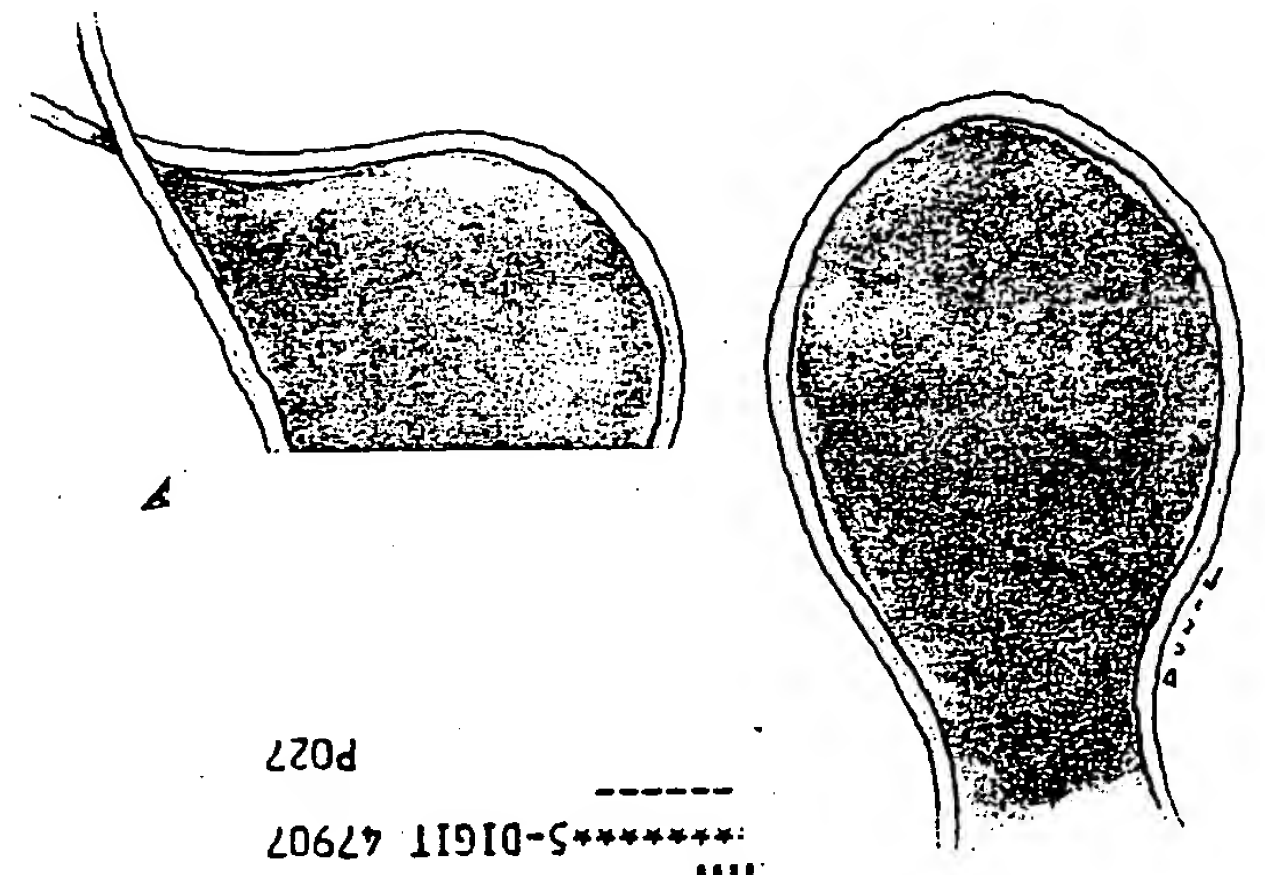
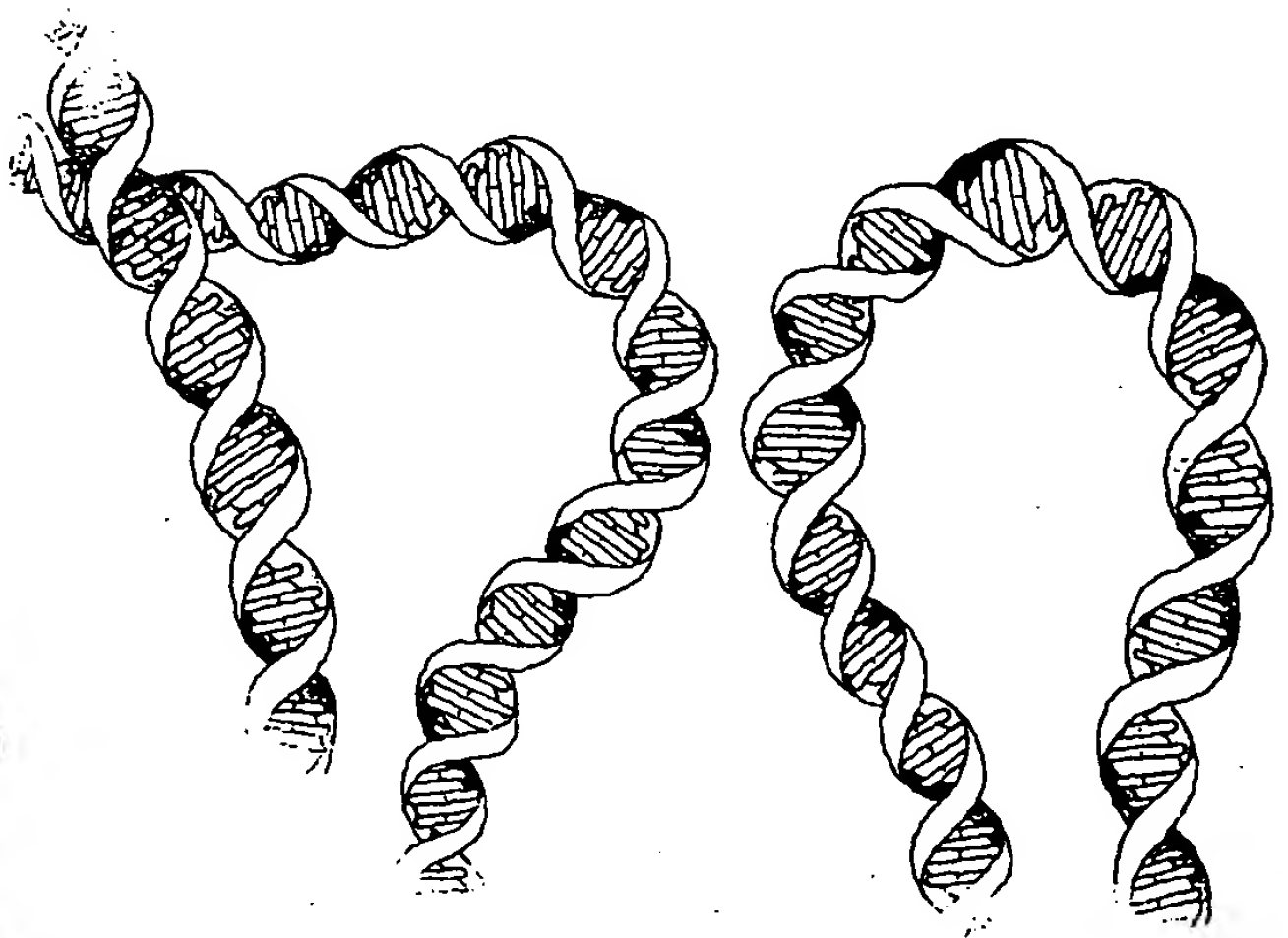
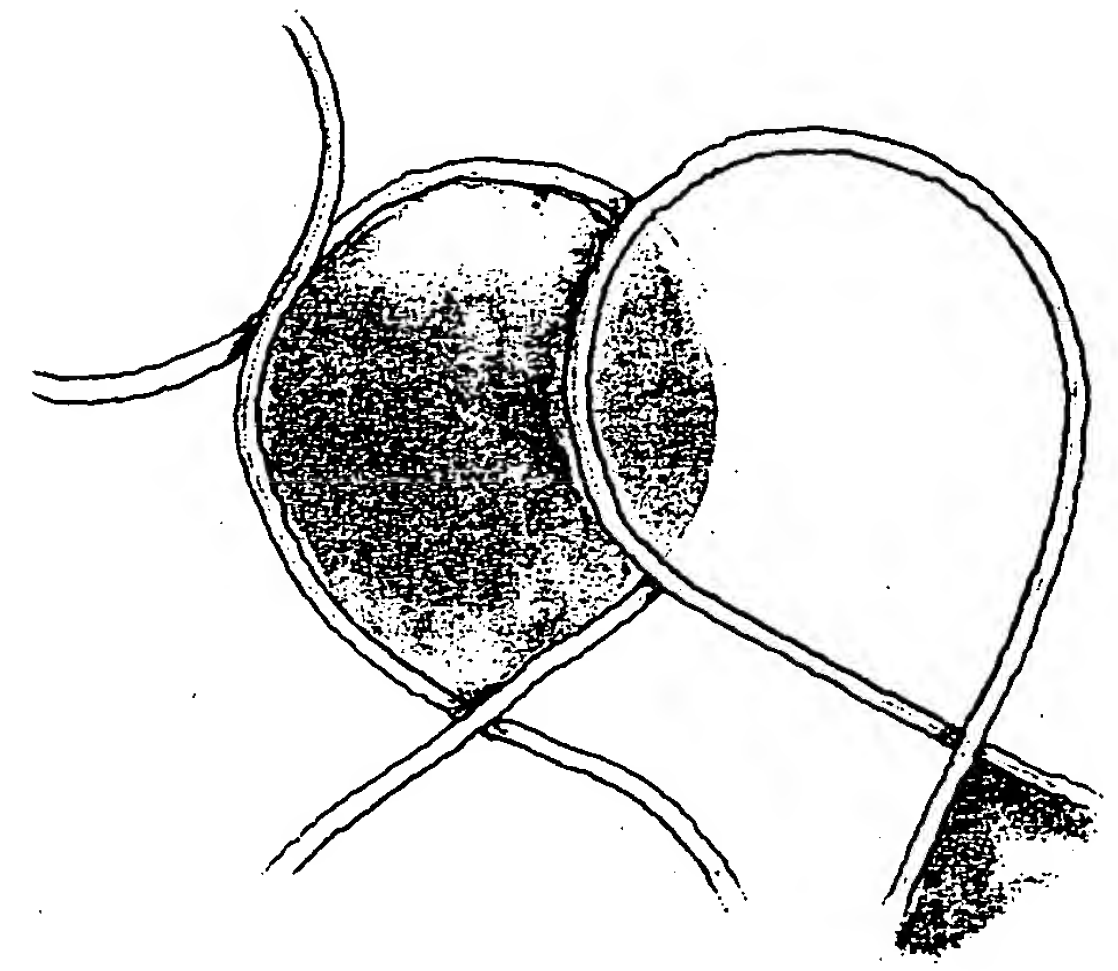
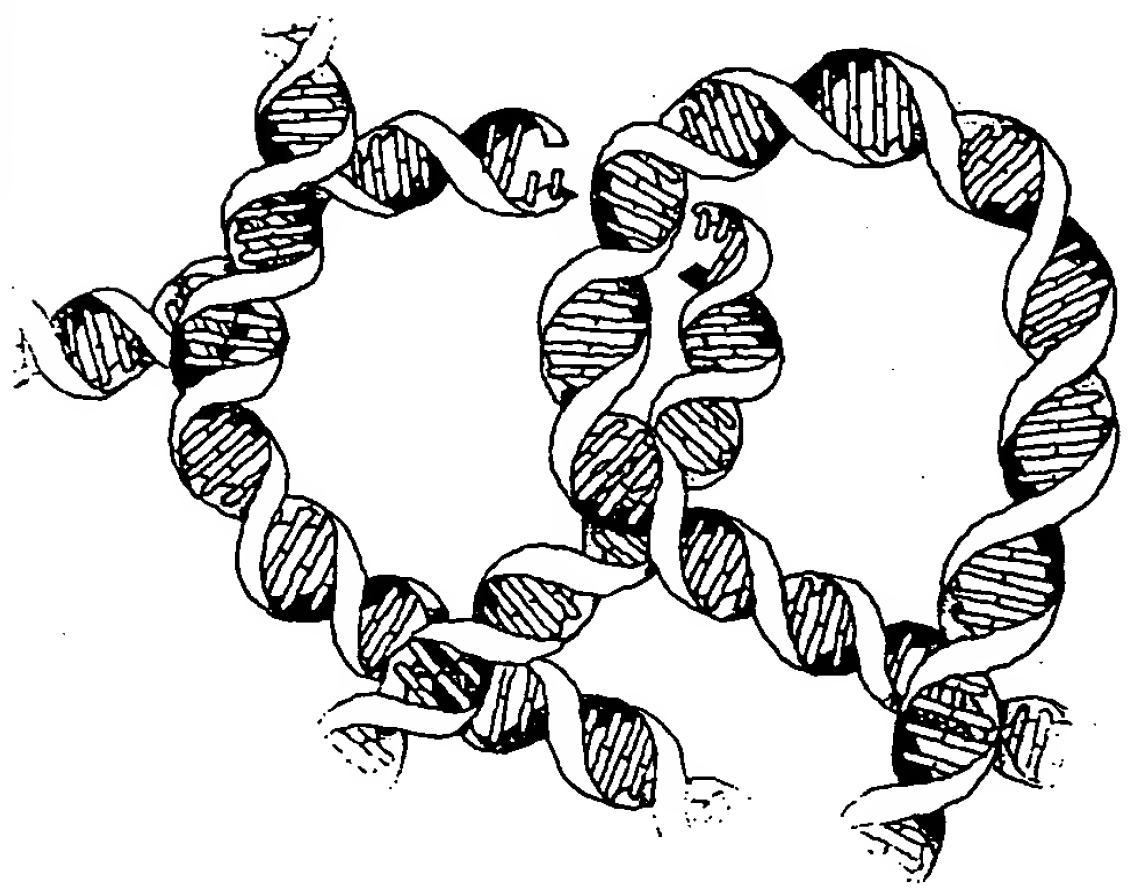
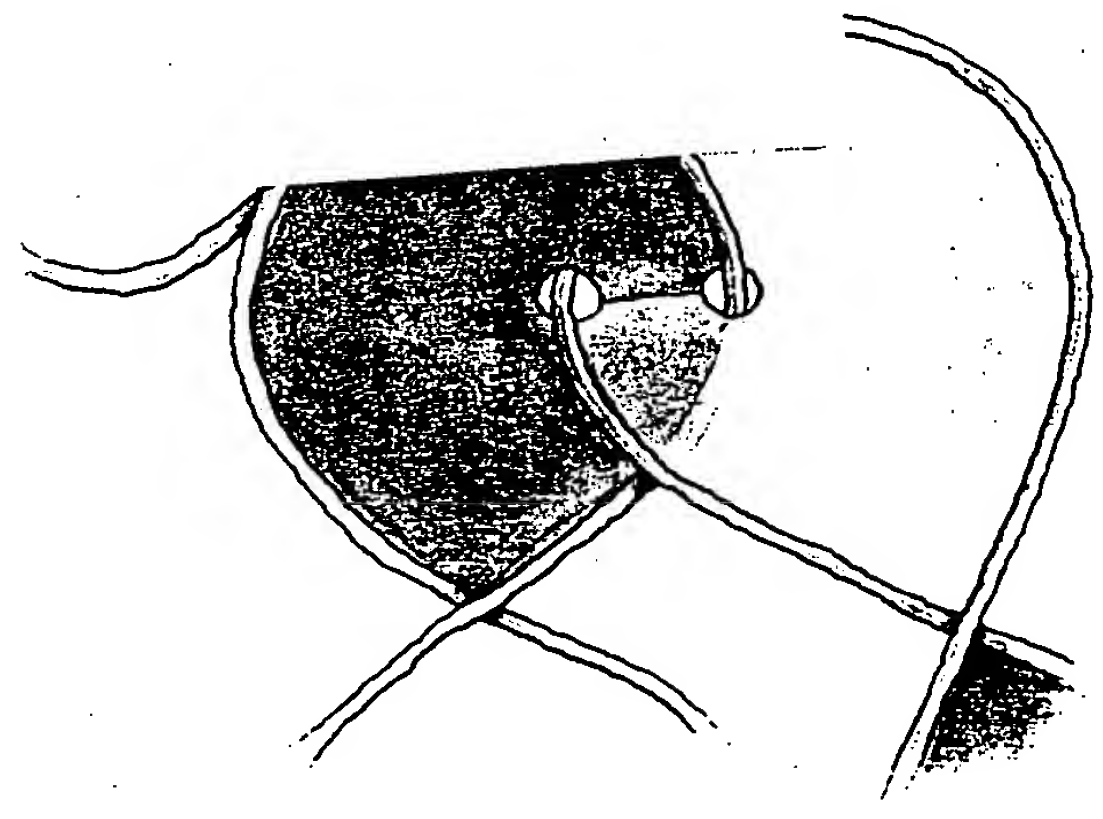
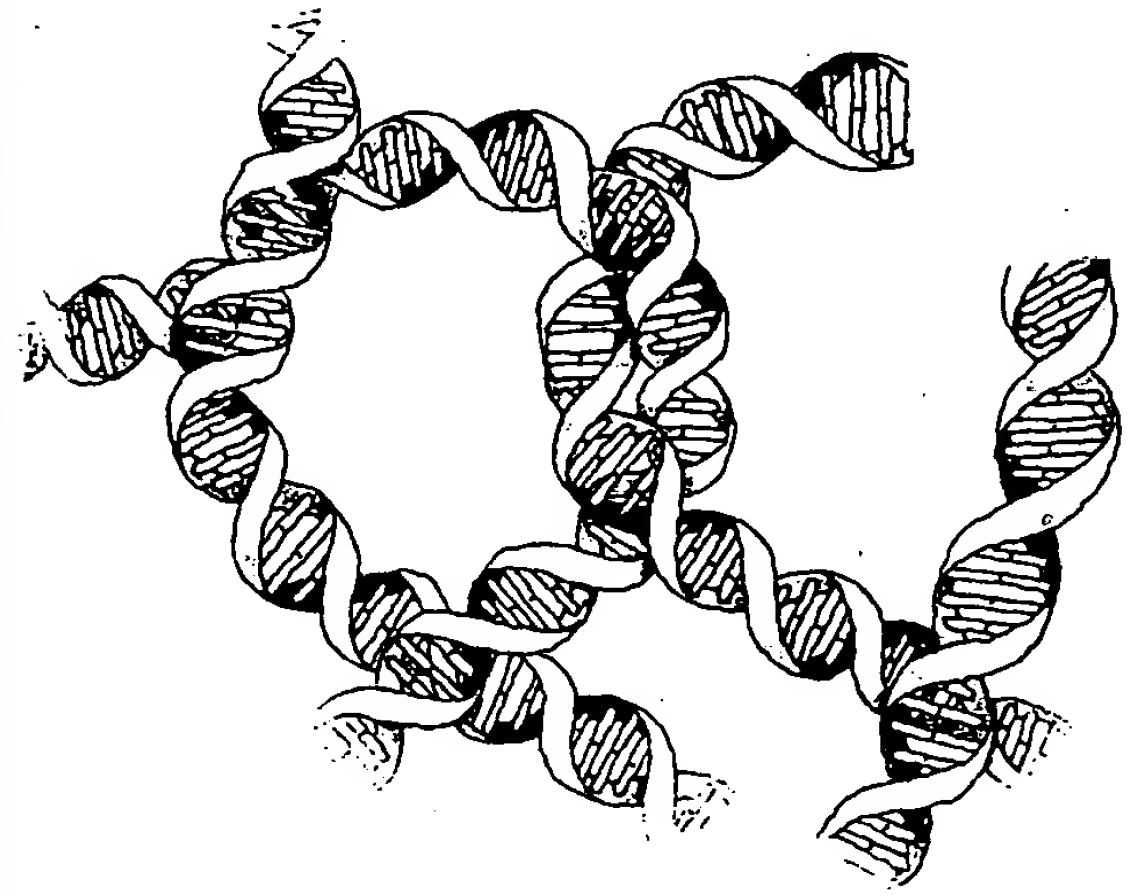
38 #13

American Scientist

JANUARY-FEBRUARY 1995

THE MAGAZINE OF THE SCIENTIFIC RESEARCH SOCIETY, SIGMA XI

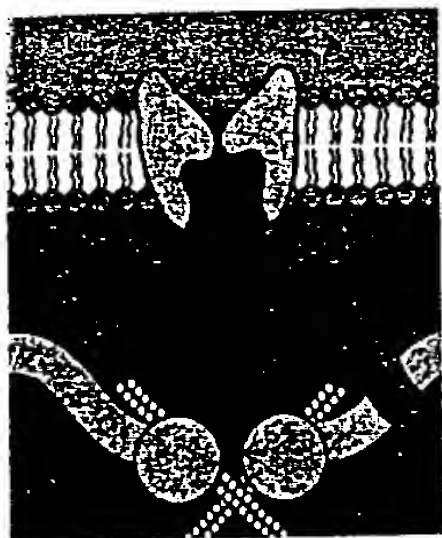
LIFE SCIENCES LIBRARY



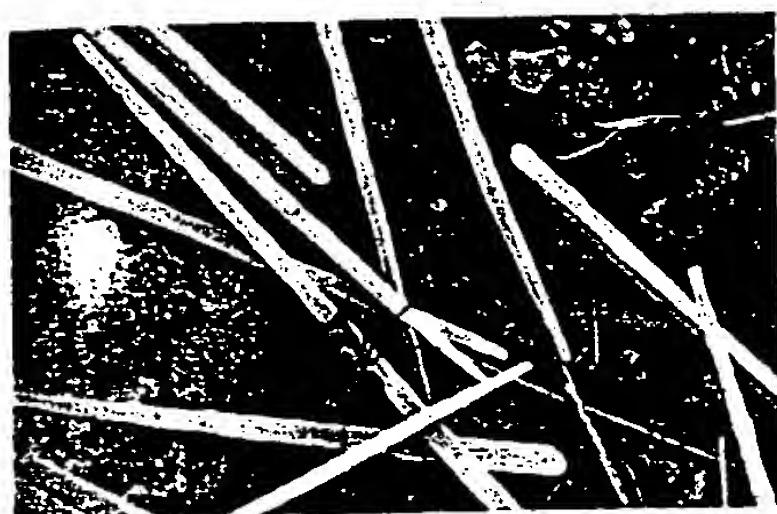
0202
*****S-DIGIT 47907
111

American Scientist

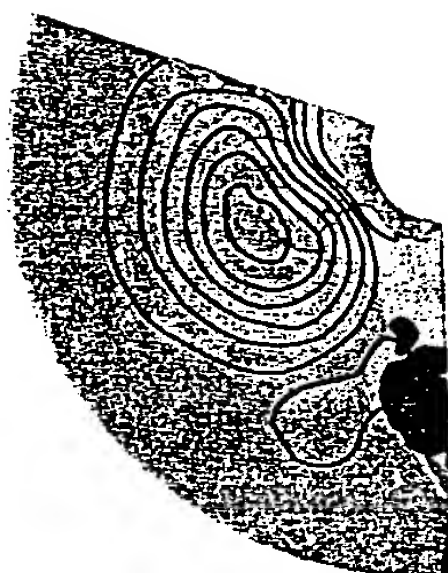
Volume 83, No. 1 January-February 1995



page 30



page 50



page 58



page 68

Departments

- 4 Letters to the Editors
- 8 **Macroscope**
Genomania and health
Ruth Hubbard
- 12 **Computing Science**
Waiting for 01-01-00
Brian Hayes
- 17 **Engineering**
Soft graphics
Henry Petroski
- 23 **Marginalia**
Classical democracy and scientific expertise
Roald Hoffmann
- 26 **Science Observer**
Mind viruses • Interacting fields
• Gone today, hair tomorrow
- 101 **Sigma Xi Today**

PURDUE UNIVERSITY

DEC 14 1994

LIBRARY

Articles

- 30 **Mechanoreceptive Membrane Channels**
Senses such as touch and hearing depend on ion traffic across the cell membrane
Owen P. Hamill and Don W. McBride, Jr.
- 38 **How Hard Is It to Untie a Knot?**
An ancient problem has been solved by modern mathematics
William Menasco and Lee Rudolph
- 50 **The Development of Organization in an Ant Colony**
Simple, local decisions can generate a complex society
Deborah M. Gordon
- 58 **The Role of Climate in Estuarine Variability**
Sifting anthropogenic trends from climate-driven fluctuations can be difficult
David H. Peterson, Daniel Cayan, Jeanne Dileo-Stevens, Marlene Noble and Michael Dettinger
- 68 **Self-Conscious Emotions**
Complex emotions develop earlier in childhood than was thought
Michael Lewis

The Scientists' Bookshelf

- 79 **Books**

The Cover

Untying a knot is a problem in both nature and mathematics. Unknotting is required whenever DNA, twisted and knotted into coils in the nucleus of the cell, prepares to replicate. DNA unknots by "cheating": An enzyme might cut one double strand, pass another through the gap and reseal the cut ends (left). In mathematics, geometric conceptions replace physical realities. In solving the ancient problem of the unknotting number, topologists have envisioned unknotting as the unhooking of "clasps" formed when one surface, bounded by a closed curve, passes through another (right). The middle and bottom configurations, distinguished by a twist in the knot, are mathematical equivalents. In "How Hard Is It to Untie a Knot?" (pages 38-49) William Menasco and Lee Rudolph show how the unknotting number has been found, with implications for biology and physics.

WAITING FOR 01-01-00

Brian Hayes

When I was a boy growing up in Toledo, Ohio, my best friend was Jimmy Liccata, whose father owned Tony's Gulf Station a few blocks away. One day Jimmy and I were snooping in the Liccata garage, doubtless up to no good, and discovered a carton of blank receipt pads from the gas station—the kind with alternating white and yellow pages, and a purple sheet of carbon paper to slip between them. The tablets were an irresistible attraction to a couple of first graders just beginning their initiation into the mysteries of the written word. I was enchanted by the magic of seeing all my scribbles reproduced in duplicate. But what I remember most vividly about those long-lost gas-station forms (imprinted with an orange-and-blue emblem that has itself disappeared from the American landscape) is the space for filling in the date. It read: “_____, 195__.” Seeing that inscription gave me my first hint of the alarming velocity of time. All my life it had been the 1950s, and up to that moment the decade had seemed absolutely endless. Now I realized that the Fifties would eventually pass away, and all that stationery would be made obsolete. The close of the decade still seemed unimaginably far off—these events took place in the summer of 1956—but even if I could not quite see myself living in the new world of 196__, I knew it was coming, inexorably. Perhaps I even foresaw that stationery imprinted “_____, 19__” would someday expire. What a thought! A new millennium.

The calendrical milestone that seemed so astronomically remote to a boy in 1956 is now bearing down on us at a sure and steady speed of 3,600 seconds per hour. We have only five years left before a certain Friday night brings to an end, all at once, a month, a year, a decade, a century and a millennium. On the following Saturday morning it is not just preprinted stationery that will become obsolete. Among all the other transformations to be expected as the new age dawns, it seems likely that many com-

puter programs will begin acting a bit strangely that day. Here are some hypothetical examples:

Shortly after midnight on January 1, 2000, a program meant to make automatic back-up copies of computer files starts replacing documents dated 01-01-00 with versions from 12-31-99 or earlier. Similarly, a programmer's tool that automatically links together the latest components of software under development begins including outdated modules.

Because many computers interpret 01-01-00 as the first day of 1900 rather than 2000, they also take it to be a Monday rather than a Saturday. As a result, traffic signals and school bells operate on their weekday schedule, and the display of arrivals and departures at the railroad station shows the Monday-morning commuter trains. Somewhere in the nation a bank is robbed because a time lock allows a vault to open on Saturday.

At the airport, all flights are canceled. A computer in the maintenance department has grounded the aircraft because they are 99 years overdue for airframe and engine overhauls. Furthermore, some pilots appear to have been on duty for 875,000 hours, in violation of union and FAA work rules.

At the local dairy, the oldest milk on hand is supposed to be shipped first, but in the early weeks of the new millennium milk from the year 00 is given precedence. Indeed, any milk remaining from December 1999 will not be scheduled for shipment until the end of 2099. Meanwhile at the bakery across town a computer calculates that bread dated 01-01-00 must be a century old, and sends it to the landfill.

The next time you open up your computerized laboratory notebook, the software informs you haughtily that all entries must have monotonically increasing time stamps. Then it accuses you of tampering with the system clock.

When you get your first bank statement of the new year, you find transactions listed in a curious nonchronological sequence, with all those from 2000 preceding those from 1999. Moreover, your \$1,000 deposit made in late December has earned almost 100 years of interest, and so you have a balance of \$400,000. Don't be too quick to spend it, however. The next day

Brian Hayes is a former editor of American Scientist. Address: 211 Dacian Avenue, Durham, NC 27701. Internet: bhayes@mercury.interpath.net.

your Visa bill arrives, and you owe \$136 million. And when the phone bill comes in, that Happy New Year call you placed just before midnight has been charged as 53 million minutes. Then the library sends you a notice of some *seriously* overdue books.

The Millennium Cruise

Events like these are awaited with anxiety—and perhaps also a hint of mischievous glee, as long as no one gets hurt—by the small band of computer professionals who track the hazards of information technology. The main gathering place for this band is the Risks Forum (1), an Internet mailing list and newsgroup moderated by Peter G. Neumann of SRI International. The foibles of computer clocks and calendars are a favorite topic in the forum. Most of the imaginary calamities mentioned above are based on speculations published in Risks over the past few years (2). Indeed, the perils of 01-01-00 have been so often anticipated in the forum that one contributor has proposed a turn-of-the-century cruise (3) for those who want to be out of harm's way when the calendar "rolls over." While the rest of us are stuck ashore, arguing over whether the 21st century begins in 2000 or 2001, the canny Risks readers will slip away aboard a craft with non-electronic controls and no computer-aided navigational equipment.

Most of the problems cited above arise from representing calendar dates with just six decimal digits, in a format that allows only two digits for the year. A program that adopts this representation is necessarily limited to a 100-year span. Most such programs interpret the years 00 through 99 as 1900 through 1999. Thus when the calendar rolls over from 99 to 00, the date does not advance into the next century but returns to 1900 again. Computers that rely on a calendar of this kind are destined to repeat the 20th century over and over; they live in a cyclic universe, where the future wraps around to become the past again.

The concepts of "before" and "after" are circular in such a world. A chronological sorting of bank transactions or batches of milk is sure to yield some strange results near the turn of the century. Similarly, the Office of Vital Statistics should not be surprised a few years from now to register an entire generation of newborn children who are older than their parents and grandparents. All these oddities are simple consequences of the arithmetic of the cyclic calendar, in which $99 + 1 = 00$, but $00 < 99$.

When the arithmetic is more complex than numeric comparison, the exact effect of calendar rollover depends on the details of how the arithmetic is done. Consider the case of a bank calculating interest on a sum of money deposited on 12-20-99 and withdrawn on 01-10-00. Subtracting 12-20-99 from 01-10-00 ought to yield not



Figure 1. The Last Judgment, as envisioned circa 1000, when the end of time seemed near. At 2000, apocalypse will be computerized. (From Georg Leidinger, *Miniaturen der Staatsbibliothek München*.)

quite -100 years, or more precisely -36,503 days. On getting such a result, one thing a computer might do is—to use the technical term—*barf*. A negative interval makes no sense in this context, and a prudently written program might well include an explicit check for this possibility; on finding a negative value, the program would stop and signal an error. Even without explicit error-checking, the negative number of days might bring the program to a halt; for example, the program might at some point attempt to take the logarithm of this number, which is an impossible operation. Or, the unexpected negative value might have just the opposite effect, causing the program *not* to halt but instead to enter an infinite loop. In all of these cases the program ultimately refuses to produce an answer, which may be the most benign failure mode available in the circumstances.

There are lots of other possible outcomes. Because a negative period of deposit is not to be expected, a programmer might write the interest-calculating procedure in such a way that it ignores the sign of the result when subtracting dates. This is the kind of arithmetic assumed in several of the hypothetical events cited above. A related but subtly different approach is to do the arithmetic with unsigned integers, a system of numbers in which negative values simply do

not exist. In one common implementation of unsigned computer arithmetic, 0 – 99 is equal to 65,437; if your bank uses this number of years in the interest calculation, it had better leave room on your statement for a 1,650-digit dollar amount.

Another possibility is that the minus sign would propagate all the way through the computation, so that you would be credited with, say, –\$397,000 in interest. This result has a certain logic to it, since the bank thinks you withdrew your money in 1900 but did not deposit it until 1999. Happily, interest on credit-card balances and loans might also cross over to the other side of the ledger, so that you would receive a fabulous refund.

In still another variation, a negative number could turn up as the exponent in the formula for compound interest. In this case both your assets and your liabilities would dwindle away to trivial sums; at –6 percent per year, a \$1,000 deposit would be reduced to \$2.95. Here's one more amusing prospect: If the bogus negative value creeps into the computation in *two* places, your interest could be either a debit or a credit depending on whether the money was on deposit for an odd or an even number of days.

And there are yet more ways for this simple-seeming calculation to go awry. Much computer arithmetic is done with 16-bit numbers, which can represent a total of 65,536 (i.e., 2^{16}) distinct values. A widely adopted convention for 16-bit arithmetic allows the integers from 0 through 32,767 to represent themselves, whereas the remaining values are interpreted as the negative integers from –32,767 through –1. The 36,503 days between 01-10-00 and 12-20-99 exceed the maximum positive value in this system and would therefore be interpreted as a negative integer, namely –29,033.

It would be easy to dream up still more error mechanisms. Indeed, it begins to appear that with enough ingenuity virtually any result could be gotten from this calculation—even the right result. Most programs on most computers will probably continue to operate normally in the new millennium. But there will also be a great blooming of bugs on that Saturday morning five years from now.

The Centenarian's Complaint

There is a quick fix for some of the difficulties noted above. Although two decimal digits can represent no more than 100 years, any 100-year span could be chosen. For example, a program could be designed to interpret the years from 30 through 99 as 1930 through 1999, yet see 00 through 29 as 2000 through 2029. In this way the program might be able to survive the millennial crisis, or at least postpone it for a few decades. But the fix has a cost: The system would become more vulnerable to other faults. In particular,

anyone whose birthdate is before 1930 might find the software distinctly unfriendly.

No matter how the dates are shifted or rearranged, a calendrical system that covers only a finite period is bound to bump up against a limit at some point. We do not even have to wait for a magic midnight such as 01-01-00 before the trouble starts. Computations done with six-digit dates already cause occasional bewilderment. For example, in 1992 Mary Bandar of Winona, Minn., was invited to join kindergarten classes when her name turned up among others identified in a database search for people born in "88"; at the time Bandar was 104 years old (4). Similarly, C. G. Blodgett's auto-insurance premium tripled after his 101st birthday, apparently because he was classified as a high-risk youthful driver (5). (There is something particularly disturbing about this story, even apart from the idea of insuring a one-year-old driver: Why did the company wait until his hundred-and-first birthday to raise the premium?) Another Risks anecdote tells of the hospital computer that interpreted the blood count of a 99-year-old man by standards appropriate to that of a newborn (6).

There are also computer systems whose built-in time bombs have a fuse shorter than 100 years. On September 19, 1989, dozens of hospitals found that computers used for bookkeeping and administration had ceased to function (7); it was not a coincidence that September 19, 1989, was the 32,768th day after January 1, 1900. A few weeks later computers running the Michigan Terminal System began to fail (8); the date was November 16, 1989, which was 32,767 days after March 1, 1900. The satellites of the Global Positioning System keep track of the date by counting the weeks since January 6, 1980. The count is maintained as a 10-bit value, and thus it has a maximum range of 1,024 weeks. It follows that on December 21, 1999, the counter will roll over, and GPS receivers will think it is 1980 all over again (9). And the grand prize for planned obsolescence goes to a personal computer sold in the mid-1980s by AT&T (10): It had a clock that ran out of ticks at the end of 1990.

Even systems that will outlast the 1900s will not get very far into the new millennium. The clock runs out on the Unix operating system in 2038, on the Macintosh in 2040 and on MS-DOS in 2048. Many other computer systems span the interval from 1901 through 2099; the likely reason for choosing these particular boundary dates is that they simplify leap-year calculations (2000 is a normal leap year, but 1900 and 2100 are not).

The Dusty Deck

Abbreviated date formats were adopted for reasons that doubtless seemed compelling at the

time. Why waste storage on digits that are always 1 and 9? Likewise, why force people to type four digits when the first two are always the same? Furthermore, some of the tools that programmers rely on encourage or even enforce a limited temporal horizon. The COBOL programming language, used for many business and financial applications, defines a year as a two-character data type. Ada, the language mandated for most Department of Defense software, is one of the 1901-to-2099 systems. Everyone knew, when these decisions were made, that time would undo them, but the day of reckoning was remote. To the programmer cobbling together a bank accounting system in 1962 it would have seemed comically pretentious to imagine that the program might still be in use in 2000. Given the brevity of the human life span, and the rapid pace of technological evolution, building a machine with a design life of 100 years should not earn you criticism for shortsightedness.

Yet here comes 01-01-00. Risks Forum readers know how the day will go. The first trouble reports will come from New Zealand, which Peter Neumann calls the "king's taster" for computer clock problems. Then the wave of failures will wash over Asia, Africa and Europe; those of us in the Americas will see it coming hours in advance, and yet we will probably be unable to do much of anything to stop it.

As computer bugs go, the problems connected with truncated and cyclic dates are not very subtle or obscure, and many of them can doubtless be fixed by simple changes. Just leaving room for four-digit years should hold us for another eight millennia. The challenge of averting computer catastrophe on 01-01-00 is not in the bugs themselves; the challenge is the "dusty-deck problem." The term comes from the era of punched cards—and so does some of the software still running today. That accounting system written in 1962 may still be at the heart of a bank's daily operations, though by now it is encrusted with thick layers of additions and patches, and no one has a clear idea of how it actually works. "Legacy systems," they call such software. Altering a legacy system in an area as fundamental as the format of dates is rather like changing a tire without stopping the car. It's a delicate operation, and now is not too soon to get started.

The prevalence of calendar-related malfunctions among the reports appearing in the Risks Forum suggests just how tricky it is to get date computations right. I can offer further evidence. In the first draft of this column I was off by 10 in my calculation of the number of days between 12-20-99 and 01-10-00. David Schoonmaker, the managing editor of *American Scientist*, caught the error and showed me, with arithmetic of convincing simplicity, how to derive the correct

answer. I had done the original calculation with a spreadsheet program, which I considered more trustworthy than my own arithmetical skills. Part of the error turned out to be the result of a careless typing mistake, but after correcting that slip, a one-day discrepancy remained. The puzzle was solved when I discovered that the spreadsheet program treats 1900 as a leap year.

The last time the calendrical odometer turned over a row of three zeroes, it was a turbulent moment in social history, with much of Christendom nervously awaiting the end of the world (11). The millennium was thought to be the time of apocalypse, the appointed Day of Judgment. I cannot suppress the idea that many were perplexed, perhaps even disappointed, when 1000 came and went, and the sun continued to rise and set so reliably. But there were no computers then.

Notes

1. The Risks Forum is distributed through the Usenet newsgroup comp.risks and is also available by electronic mail, either through the BITNET LISTSERV mechanism (send the message SUBSCRIBE RISKS) or by requesting a subscription from risks-request@csl.sri.com. An archive of the forum is available through the anonymous ftp protocol from crvax.sri.com. The archive can be searched by means of the WAIS protocol, using the risks-digest.src database on the WAIS server cmns-moon.think.com. Excerpts from the forum appear in *Software Engineering Notes*, the quarterly journal of the ACM Special Interest Group for Software Engineering, and in the *Communications of the ACM*. The *Communications* excerpts of January 1991 (Vol. 34, No. 1, p. 170) concern clocks and calendars. Much material from the forum, with additional analysis and commentary, is collected in a recent book: Peter G. Neumann, 1995, *Computer-Related Risks*, New York: The ACM Press and Reading, Mass.: Addison-Wesley Publishing Company. See pp. 85-92 for a discussion of clock and calendar problems.
2. See especially: Paul Robinson. The danger of six-digit dates. Risks Forum 16:32, August 15, 1994.
3. Steve Peterson. Re: Turn of the century date problems. Risks Forum 14:45, March 29, 1993.
4. Ed Ravin. Call for the class of '88. Risks Forum 14:44, March 29, 1993.
5. Lee F. Breisacher. Risk of aging. Risks Forum 4:9, November 10, 1986.
6. David B. Benson. Another 100-year computer saga. Risks Forum 9:73, March 6, 1990.
7. Joe Morris. Hospital problems due to software bug. Risks Forum 9:26, September 20, 1989. Will Martin. Re: Hospital problems due to software bug. Risks Forum 9:27, September 21, 1989. Steve VanDevender. Re: Hospital problems due to software bug. Risks Forum 9:28, September 24, 1989.
8. Brian Randell. Another foretaste of the millennium. Risks Forum 9:45, November 17, 1989. Also corrigendum November 21, 1989.
9. Marc Auslander. Not enough bytes bites again. Risks Forum 16:48, October 21, 1994. Dave Moore. Re: Not enough bytes bites again. Risks Forum 16:49, October 24, 1994.
10. Daniel J Yurman. Re: AT&T machines and dates. Risks Forum 13:05, January 21, 1992.
11. Henri Focillon. 1969. *The Year 1000*. New York: Frederick Ungar Publishing Co.

#39

#13

Excel 4 for Windows Made Easy

Martin S. Matthews

Osborne McGraw-Hill

Berkeley New York St. Louis San Francisco
Auckland Bogotá Hamburg London Madrid
Mexico City Milan Montreal New Delhi Panama City
Paris São Paulo Singapore Sydney
Tokyo Toronto

Osborne McGraw-Hill
2600 Tenth Street
Berkeley, California 94710
U.S.A.

For information on translations or book distributors outside of the U.S.A.,
please write to Osborne McGraw-Hill at the above address.

Excel 4 for Windows Made Easy

Copyright © 1992 by Martin S. Matthews and Carole Boggs Matthews. All rights reserved. Printed in the United States of America. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

1234567890 DOC 998765432

ISBN 0-07-881807-9

Information has been obtained by Osborne McGraw-Hill from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, Osborne McGraw-Hill, or others, Osborne McGraw-Hill does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from use of such information.

Author: Martin S. Matthews

Title: Excel 4 for Windows Made Easy

Copyright 1992

Publisher: Osborne McGraw-Hill, Berkeley, California

Pages 343-387

Material is reproduced with permission of The McGraw-Hill Companies (February, 2000)

10

Dates, Functions, and Macros

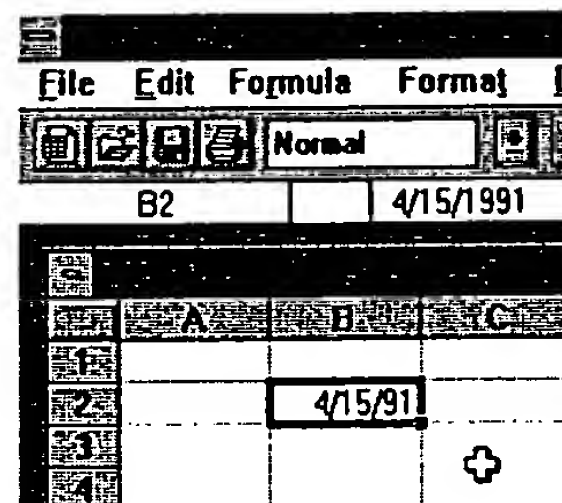
This chapter brings together three subjects that do not fit easily into any other category: using dates and times, functions, and macros. All three subjects need a forum of their own, and they are the finishing touch—the “icing on the cake” that makes a good worksheet a great worksheet.

Using Dates and Times

From schedules to dates on reports and time-related financial calculations, dates and times are important aspects of the problems Excel addresses. The sections that follow discuss how Excel handles dates and times internally, various ways Excel formats dates and times, and the date and time functions and arithmetic.

Dates and Excel

Dates do not form a nice, neat, linear progression. You cannot add 12 days to April 28th and get May 10th without knowing how many days there are in April. Microsoft has solved this problem by establishing a date serial number scheme. This scheme allocates one number for every day from January 1, 1900 (date serial number 1) to December 31, 2078 (date serial number 65380). Microsoft also provides formatting and formulas to convert the date serial number to a specific calendar date. Internally Excel uses the unformatted date serial number. You can format the date serial number in several ways and get a normal-looking calendar date from a serial number. For example, when formatted with the first Excel date format, the serial number 33343 becomes 4/15/91, as shown here:



In other words, simply typing **33343**, pressing **(ENTER)**, and formatting the number with Format Number m/d/yy produces the date 4/15/91. The following instructions demonstrate several other dates. Your computer should be on, Excel should be loaded, and you should have a blank worksheet on your screen.

1. Select B2:F2, choose Number from the Format menu, select the Date category, the m/d/yy format, and click on OK. The range B2:F2 is formatted as m/d/yy dates.
2. Type **33343** and press **(RIGHT ARROW)**. The date 4/15/91 appears in B2.
3. Type **30638** and press **(RIGHT ARROW)**. The date 11/18/83 appears in C2, as shown in the following illustration:

	A	B	C	D	E
1					
2		4/15/91	11/18/83		
3					+
4					

4. Type 1, press **(RIGHT ARROW)**, type 65380, and press **(ENTER)**. The date 1/1/00 appears in D2, and E2 contains 12/31/78.

Note that when you are working in the next century, the two-digit year format can be confusing. Change the format next. E2 should still be the active cell.

5. Choose Number from the Format menu, click after the last “y” in the Code text box, type yy, and click on OK. E2 fills with #s because the date is now too big for the cell.
6. Drag on the intersection of columns E and F in the heading for about one tenth of an inch to widen the column. The date 12/31/2078 appears in E2, as shown here:

	A	B	C	D	E	F
1						
2		4/15/91	11/18/83	1/1/00	12/31/2078	
3						
4						

There is one abnormality in Microsoft's date scheme. The year 1900 was not a leap year, even though the year was evenly divisible by four. Therefore, Excel assigns a date serial number to February 29, 1900, which didn't exist. The only impact of this is that date arithmetic spanning February 28, 1900 through March 1, 1900 is off by one day. All date serial numbers and calculated dates from March 1, 1900 onward are correct.

Times and Excel

Microsoft has also developed a scheme for calculating time: the time is added to the date serial number as a decimal fraction of a 24-hour day.

Therefore, midnight is 0.000000, noon is 0.500000, and 11:59:59 PM is 0.999988. When the decimal fractions are formatted with Excel as times, they produce normal-looking time numbers on either a 12- or 24-hour basis. The following steps show how several times are entered.

1. Select B2:E2, press **(DEL)**, click on All, and click on OK to erase both the contents and formats of B2:E2.
2. Choose Number from the Format menu, select the first time format, h:mm AM/PM, and click on OK. The range B2:E2 is formatted with the first time format.
3. Type .65 and press **(RIGHT ARROW)**. Cell B2 contains 3:36 PM, as shown here:

Sheet1					
	A	B	C	D	E
1					
2		3:36 PM			
3					

4. Type .45 and press **(RIGHT ARROW)**. Cell C2 contains 10:48 AM, as shown here:

Sheet1					
	A	B	C	D	E
1					
2		3:36 PM	10:48 AM		
3					

5. Type 0, press **(RIGHT ARROW)**, type .9999, and press **(ENTER)**. Cells D2 and E2 contain 12:00 AM and 11:59 PM, as shown here:

Sheet1						
	A	B	C	D	E	F
1						
2		3:36 PM	10:48 AM	12:00 AM	11:59 PM	
3						

Dates and times are stored in one number. For example, 3:36 PM April 15, 1991, is stored as 33343.65. In a single cell you can display this as a date, as a time, or both, depending on the formatting.

Formatting Dates and Times

There are four date formats, four time formats, and one combined date and time format built into Excel. The date formats use the integer part of a date serial number, and the time formats use the decimal part. The decimal part of a number is ignored by a date format, and the integer part of a number is ignored by a time format. If a date format encounters a number that is negative or greater than 65380, the cell fills with #s. If a time format encounters a negative number, the cell fills with #s.

The nine built-in date and time formats are shown in Figure 10-1. You can, of course, make your own. The components needed to construct your own formats are shown in Chapter 6. Some examples of custom date and time formats are shown in Figure 10-2.

Figure 10-1. Built-in date and time formats

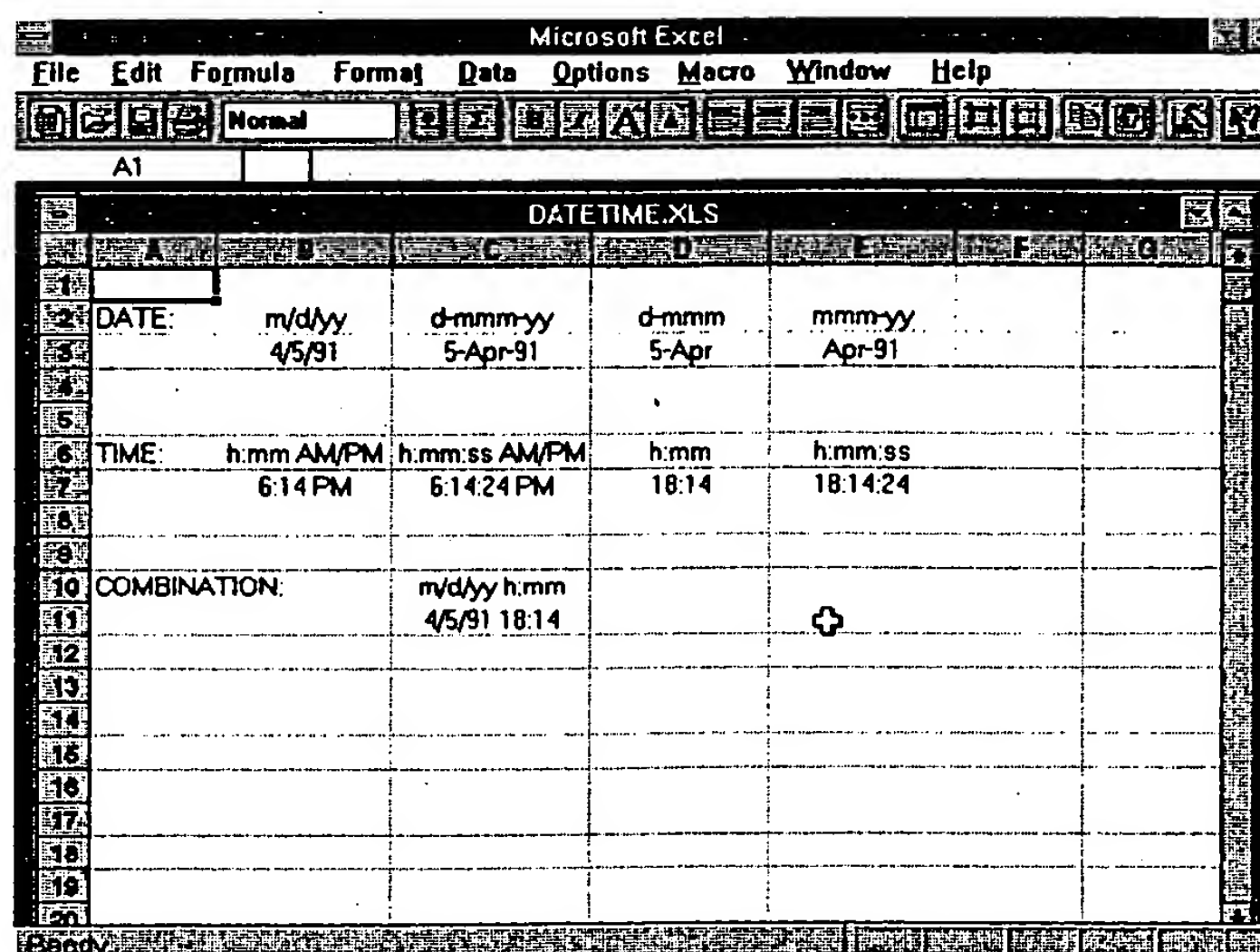
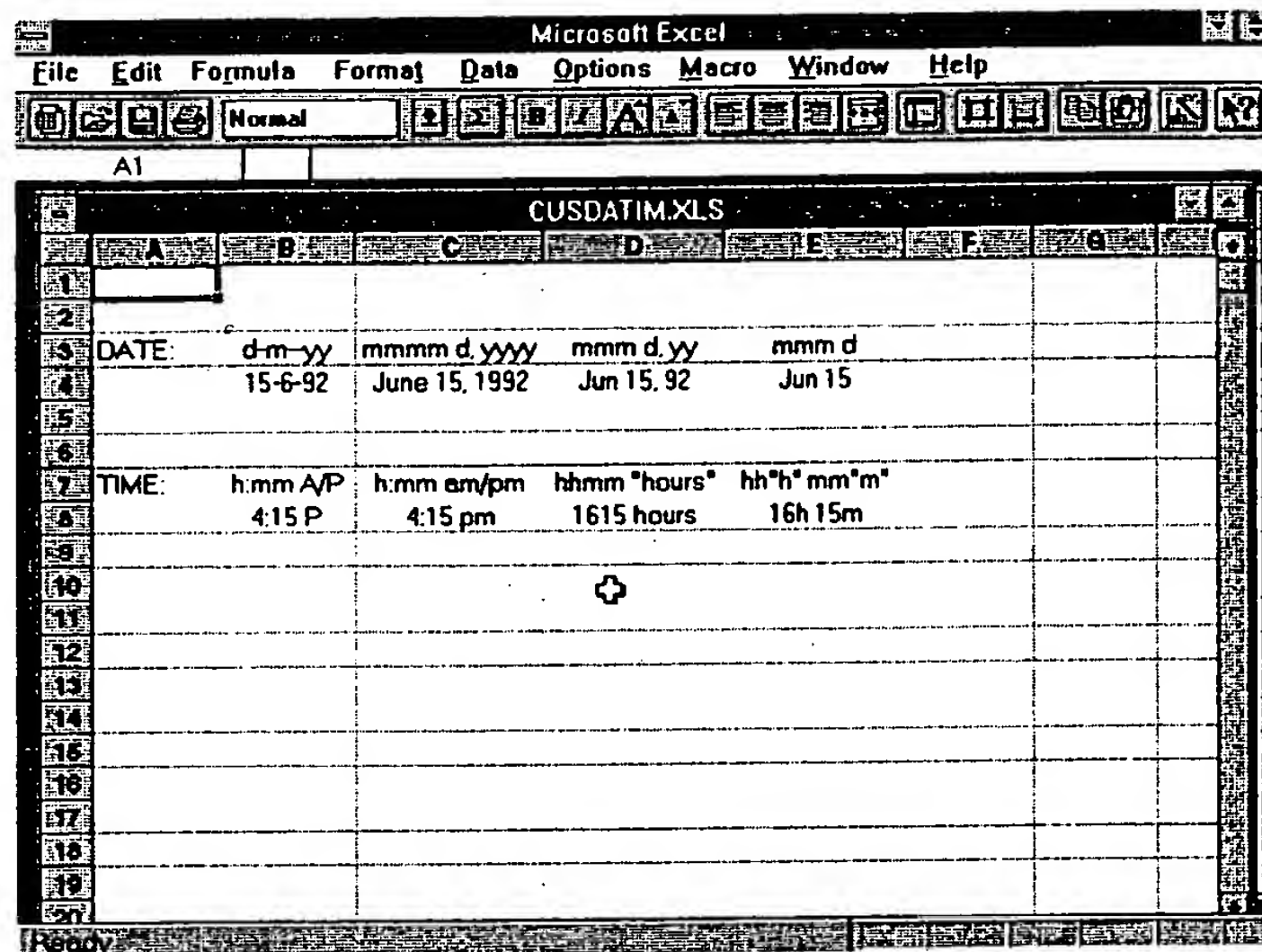


Figure 10-2. *Examples of custom date and time formats*



The choice of which format to use is one of personal taste. Some formats require wider columns than others, which may have some bearing on your decision. Also, there is no reason you cannot mix formats in a worksheet.

Date and Time Functions

Date and time functions use the date serial number to calculate various date- and time-related numbers. There are seven date and five time functions that are used in date and time arithmetic. In addition there are two functions that produce the current date and/or time. Of the fourteen functions, five produce a date or time serial number that must be formatted in order to be displayed properly. Any of the formats may be used with the functions.

Date Functions

The seven date functions are shown in Figure 10-3. The first two produce date serial numbers, shown in the middle column of Figure 10-3, that can be formatted as dates, shown in the third column. The next four functions

Figure 10-3. Date functions

	A	B	C	D	E
2		=DATE(91,4,15)		33343	4/15/91
3		=DATEVALUE("4/15/91")		33343	4/15/91
4		=DAY(33343)		15	15
5		=MONTH(33343)		4	4
6		=WEEKDAY(33343)		2	2
7		=YEAR(33343)		1991	1991
8		=DAYS360("4/15/91","4/15/92")		360	360
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

transform a date serial number into part of a date. The last function calculates the number of days between two dates using a 360-day year.

The DATE function takes three integers—one for the year, one for the month, and one for the day—and computes the date serial number. As with all functions, the three arguments (year, month, and day) can be integers that are entered directly into the function, or they can be addresses or range names that refer to cells containing or computing integers suitable for the function. DATE is used when you break out a date to sort or to use as a database criterion and you want to display the date that results from the combined pieces.

DATEVALUE converts a date in text form to a date serial number. DATEVALUE looks for text as an argument. Therefore, a date that is directly entered into the function must be enclosed in quotation marks, as shown in Figure 10-3.

The next four date functions, DAY, MONTH, WEEKDAY, and YEAR, perform the opposite function of DATE: they split out the day, month, weekday, or year from the date serial number. WEEKDAY returns an integer from 1 for Sunday to 7 for Saturday.

Time Functions

The five time functions are shown in Figure 10-4. The first two produce the date serial numbers shown in the middle column of Figure 10-4, which can be formatted as the times shown in the third column. The last three functions transform a time serial number into the components of time.

The TIME function uses three integers—one for hours, one for minutes, and one for seconds—to compute the time serial number. TIME is used when you break out a time for sorting or for use as a database criterion and you want to display the time that results from the combined pieces.

You use TIMEVALUE to convert a time that is entered as a label to a date serial number. TIMEVALUE looks for a label as an argument. Therefore, a time that is directly entered into the function must be enclosed in quotation marks, as shown in Figure 10-4.

The next three date functions, HOUR, MINUTE, and SECOND, perform the opposite function of TIME: they split out the hour, minute, or second from the time serial number.

Current Date and Time Functions

The functions used to produce the current date and/or time are NOW and TODAY. Examples of their use are shown in Figure 10-5. These functions use the internal clock-calendar in your computer to determine the current date and time. NOW produces both the integer and decimal components needed for both date and time display. If you are using the current date in a formula or with another function, use TODAY to obtain the integer part of the current date. The decimal (time) part of NOW can cause inaccuracies in date calculations. If you simply are displaying the date, NOW by itself works. Date formats ignore the decimal part of the number.

Entering and Generating Dates and Times

You have just seen how you can enter dates and times either by entering the date serial number (which is not very practical because you don't know what it is in most instances) or by using one of the functions that convert a date or time to the date serial number. Also, you have seen how to generate the current date and/or time. You have two other ways to get dates and times into Excel. First, just typing a date or time on the worksheet in an Excel format

Figure 10-4. Time functions

The screenshot shows the Microsoft Excel interface with the following data in the spreadsheet:

	A	B	C	D	E	F
1						
2						
3			=TIME(11,15,20)	0.468981		11:15:20 AM
4			=TIMEVALUE("11:15:20 AM")	0.468981		11:15:20 AM
5						
6						
7			=HOUR(468981)	11		11
8						
9			=MINUTE(468981)	15		15
10						
11			=SECOND(468981)	20		20
12						
13						
14						
15						
16						
17						
18						
19						
20						

Figure 10-5. Current date and time functions

The screenshot shows the Microsoft Excel interface with the following data in the spreadsheet:

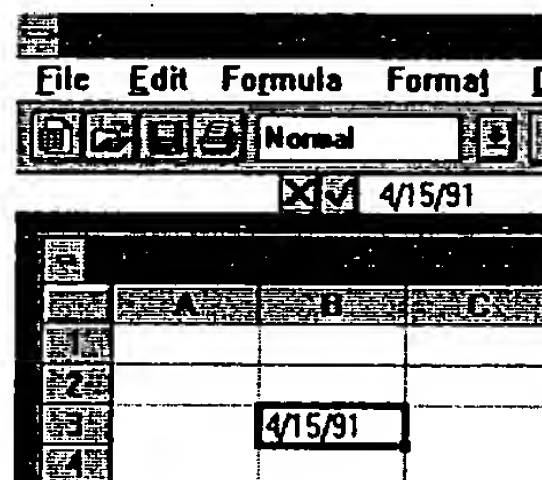
	A	B	C	D	E	F
1						
2						
3			=NOW()	33671.50705		3/8/92 12:10
4						
5						
6			=TODAY()	33671		3/8/92 0:00
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						

produces a date or time serial number. Second, the Data Series option generates a sequence of dates or times.

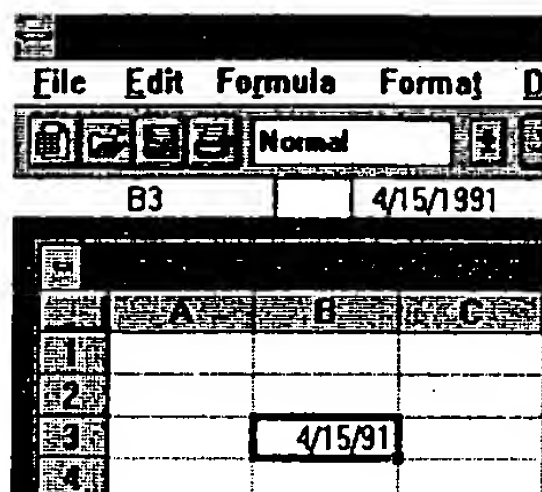
Direct Entry of Dates and Times

You can enter a date or time directly into Excel in any recognized format and get a date or time serial number. It does not have to be one of the built-in formats. When you enter a date or time, it is automatically formatted as the first date or first time format, respectively. The following instructions provide some examples of direct entry of dates and times:

1. Choose New from the File menu, and click on OK for a new worksheet.
2. Move the active cell to B3.
3. Type 4/15/91, and it goes in the edit area, as shown here:



4. Press (ENTER). The date, 4/15/91, is converted to 33343, and the cell is automatically formatted as m/d/yy, as shown here:

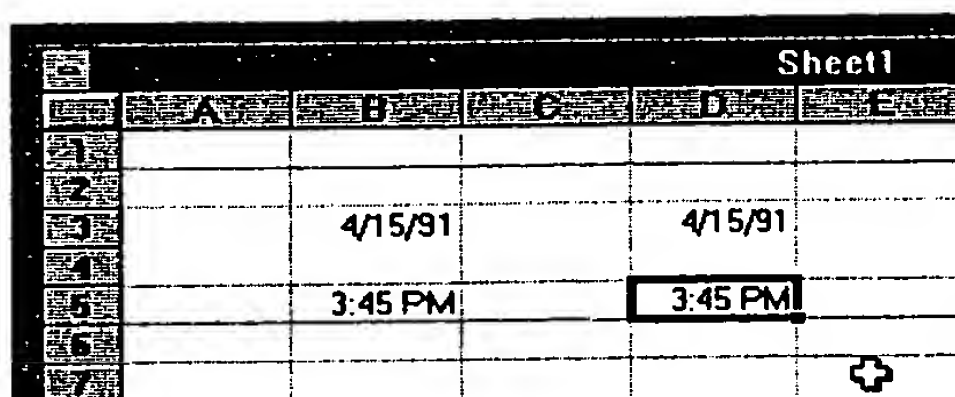


If you would like to see that 4/15/91 is in fact the date serial number 33343, reformat B3 with the General format. You might also want to do this when you get to time values.

5. Click on D3, type 4-15-91, and press **(ENTER)**. The date is converted to 33343 and automatically formatted as 4/15/91 even though 4-15-91 is not a built-in format.

If you want to enter a formula that looks like a date, you must put an equal sign in front of it.

6. Click on B5, type 3:45 pm, and press **(ENTER)**. The time, 3:45 PM, is converted to .65625, and the cell is automatically formatted with h:mm AM/PM.
7. Click on D5, type 3:45 p, and press **(ENTER)**. The time, 3:45 p, is converted to .65625, and the cell is automatically formatted as 3:45 PM even though 3:45 p is not a built-in format, as shown here:



The screenshot shows a portion of an Excel spreadsheet with columns A through E and rows 1 through 7. The title bar at the top indicates 'Sheet1'. The data is as follows:

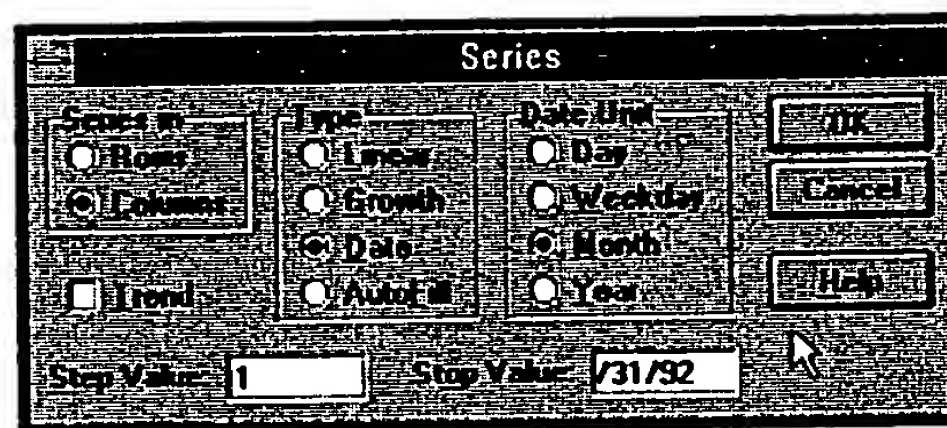
	A	B	C	D	E
1					
2					
3		4/15/91		4/15/91	
4					
5		3:45 PM		3:45 PM	
6					
7					

Generating a Series of Dates

In Chapter 8, you saw how you can generate dates with the Data Series option, which is a very capable and flexible tool. From any starting date to any ending date within the 178-year range of Excel's date scheme, you can generate as many dates as you can hold in the memory of your computer. If you are generating dates, you can increment them by a number of days, weekdays, months, years, or fractions thereof.

The following steps give several examples of generating dates and times with the Data Series option:

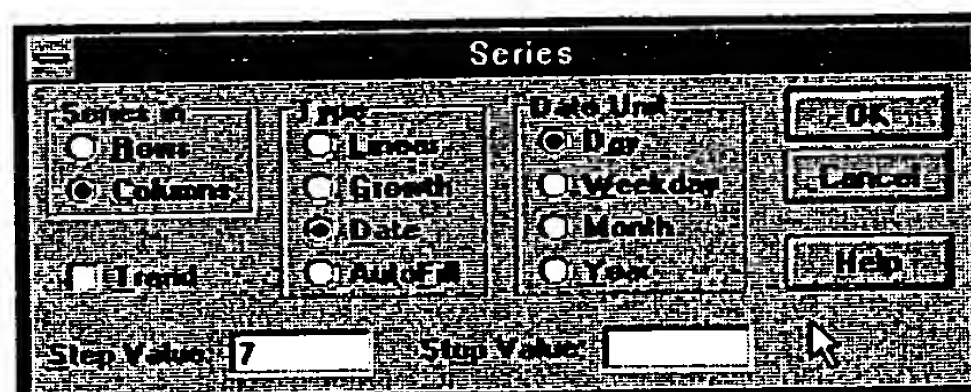
1. Choose New from the File menu and click on OK to create a new worksheet.
2. Select B2:F13, choose Number from the Format menu, select Date, the m/d/yy format, and click on OK to format the selected area.
3. Click on B2, type 1/31/92, press **(ENTER)**, select B2:B13, and choose Series from the Data menu. In the Series in and Type fields, Columns and Date should already be selected. Select Month for the unit and type 12/31/92 as the Stop value. One month is the default Step value, as shown here:



4. Press **(ENTER)**. B2:B13 fills with a series of dates that are one month apart, from 1/31/92 through 12/31/92.

This series provides the actual month end, 1/31, 2/29, 3/31, 4/30, and so on, not just 30- or 31-day intervals. It can be a very useful capability.

5. Click on D2, type 1/31/92, press **(ENTER)**, select D2:D13, choose Series from the Data menu, and type 7 for the Step value. Seven days or one week is the intended step value, as shown here:



- When you are done, your screen should look like Figure 10-6. Column B shows a progression by month, column D shows a progression by week, and column F shows a progression by year.

Generating times with Excel is not much different from generating dates. While there are no ready-made increments like hours, minutes, and seconds, you can use the standard time notation of hh:mm:ss to indicate a step value.

[illegible]

For example, a step value of one second would be 00:00:01, one minute would be 00:01:00, and one hour would be 01:00:00.

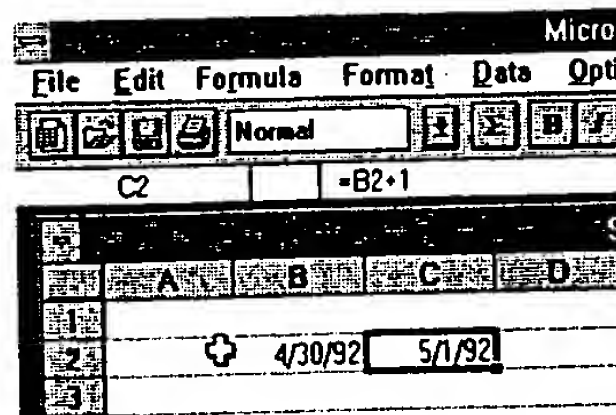
The next set of steps demonstrates several data series that produce times:

1. Choose New from the File menu, and click on OK to create a new worksheet.
2. Select B3:F15, choose Number from the Format menu, select Time, h:mm:ss AM/PM, and click on OK. The selected range is formatted with the second time format.
3. Choose Column Width from the Format menu, type 12 for the new width, and click on OK. Columns B through F widen to 12 to handle the full time format.
4. Click on B3, type 11:00:00, press **(ENTER)**, select B3:B15, choose Series from the Data menu, type 00:00:01 as the step value, and press **(ENTER)**. B3:B15 fills with a series of times that are one second apart, from 11:00:00 through 11:00:12.
5. Click on D3, type 11:00:00, press enter, select D3:D15, choose Series from the Data menu, type 00:01:00 as the step value, and press **(ENTER)**. D3:D15 fills with a series of times that are one minute apart, from 11:00:00 through 11:12:00.
6. Click on F3, type 11:00:00, press **(ENTER)**, select F3:F15, choose Series from the Data menu, type 01:00:00 as the step value, and press **(ENTER)**. F3:F15 fills with a series of times that are one hour apart, from 11:00:00 AM through 11:00:00 PM.

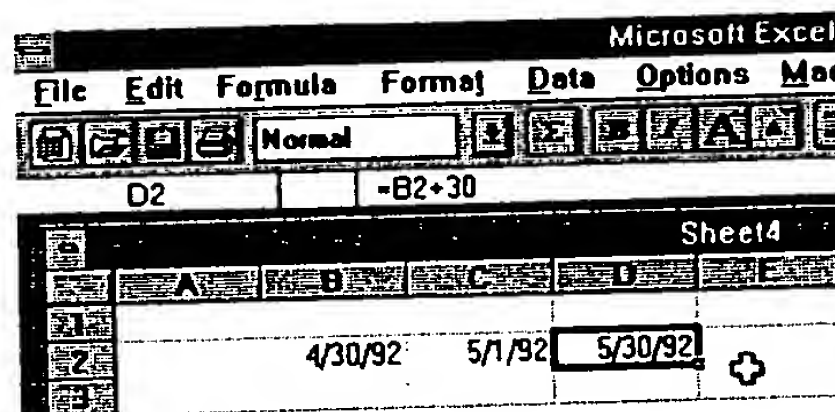
When you are done, your screen should look like Figure 10-7. Column B shows a progression by second, column D shows a progression by minute, and column F shows a progression by hour, all formatted for a 12-hour clock.

Date and Time Arithmetic

One of the primary reasons Microsoft developed the date serial number was to allow easy date and time arithmetic. For example, you can add 1 to a date and get the day following, as shown in the next illustration:

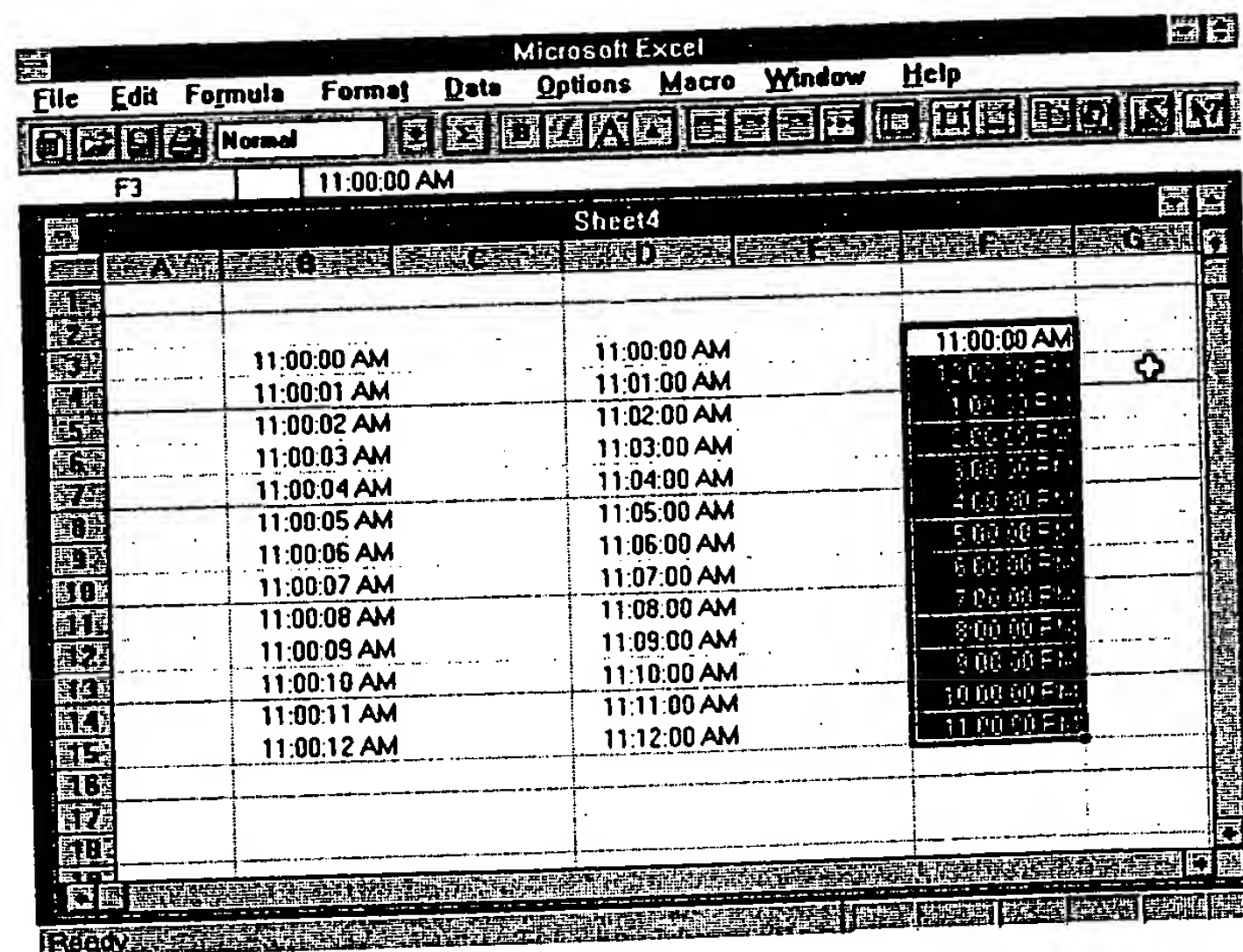


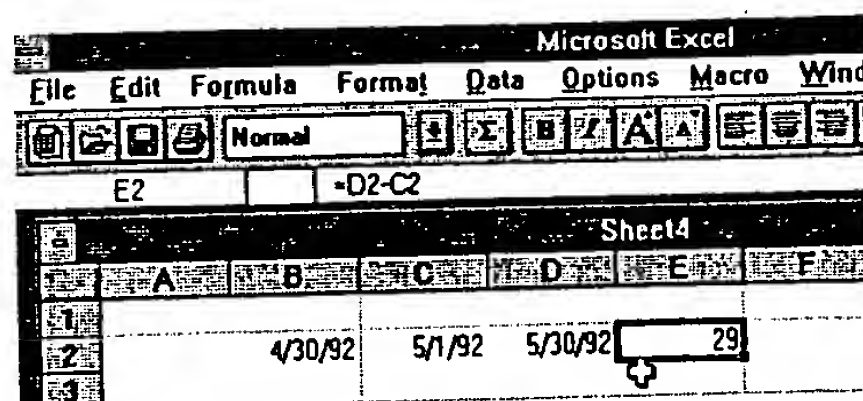
You can add 30 days and get the appropriate day in the next month:



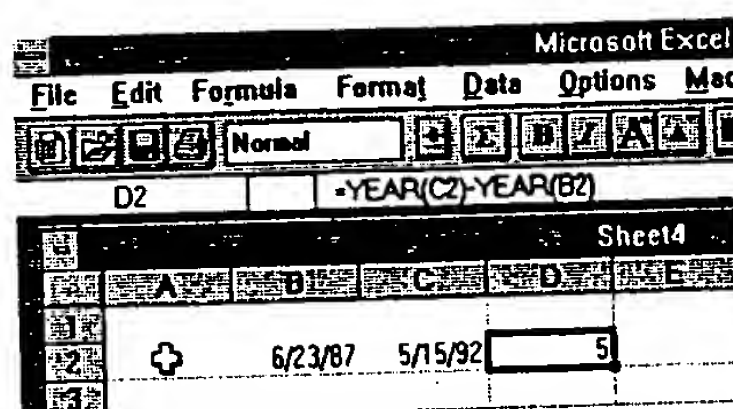
You can also subtract two dates and get the number of days between them.

Figure 10-7. Time series by second, minute, and hour

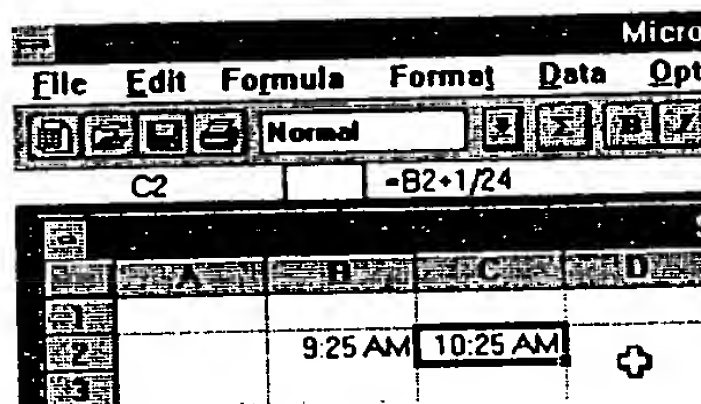




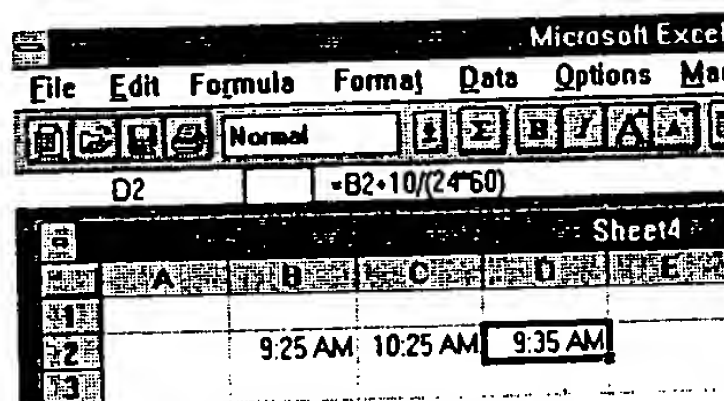
The date functions are useful in date arithmetic. For example, you can use YEAR to determine the number of years between two dates, as shown here:



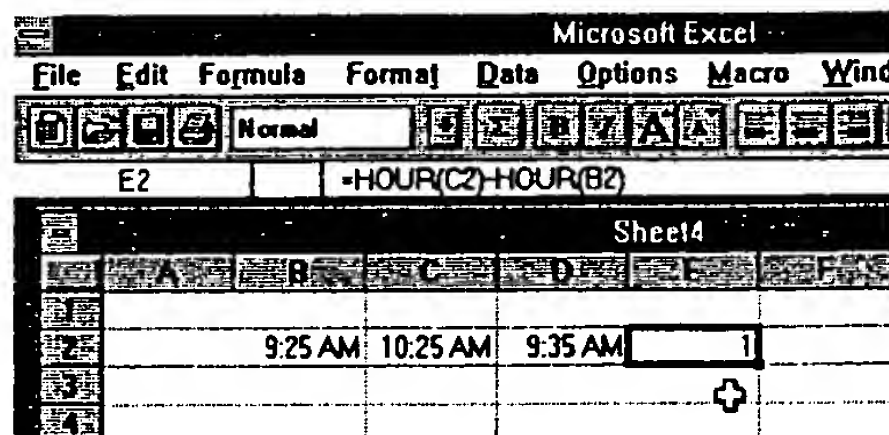
Time arithmetic is a little more complex in that you must add fractions. For example, to add one hour, you must add 1/24th, as shown here:



Adding ten minutes requires the fraction 10/(24*60):



Time functions are useful in time arithmetic. For example, you can determine the number of hours between two times with two HOUR functions, as shown here:



Functions

In earlier chapters you gained some familiarity with statistical functions, database statistical functions, and lookup functions, and you just learned about date and time functions in this chapter. Excel has five other types of functions: financial, informational, logical, mathematical (including matrix and trigonometric), and text. These are discussed here, but first let's look at how functions are used and created.

Using Functions

Functions are ready-made formulas. They perform a previously assigned task that usually involves a calculation but may also include a nonarithmetic operation. Functions always produce a result in the cell in which they are entered. For example, SUM produces a value that is the arithmetic addition of a set of numbers, and UPPER produces a text string that is all uppercase. Functions provide a faster way to accomplish many tasks. For example, using SUM(*range*) is quicker than adding each individual cell in the range if the range contains three or more cells. Functions are the only way some tasks can be accomplished. For example, using NOW or TODAY are the only ways you can read your computer's clock-calendar with Excel.

Specifying Arguments

Many functions require pieces of information to perform their task, for example, the range in the SUM(*range*) function. These pieces of information are called *arguments*. The number of arguments in a function varies between 0 and 14, and the length of arguments in a function is limited to 255 characters, including any quotation marks.

Arguments can be numbers, text, arrays, references, and logical or error values, as outlined here:

- Numbers used as arguments in a function can be numerals, numeric formulas, or addresses or range names for cells that contain numbers or numeric formulas.
- Text is any sequence of letters, numbers, spaces, or symbols. Text in a function can be literal text enclosed in quotation marks, a text formula, or an address or range name for a cell that contains literal text or a text formula.
- An array is a rectangular set of values (a range) that is treated in a special way. An array is enclosed in braces ({ }) and has a semicolon between rows. For example, the array {5,6,7;3,4,5;1,2,3} is a 3-by-3 array, with three rows that each contain three columns. Arrays used as arguments in a function can be entered directly, result from a formula that evaluates to an array, or be a set of addresses or range name for a range that contains an array or a formula that evaluates to an array.
- References can be addresses, range names, or any formula that evaluates to an address or range name.
- A logical value is either True or False. You can enter True and False in either upper- or lowercase letters, but Excel converts them to uppercase. In a function, logical values may be entered directly, result from a logical formula that evaluates to either True or False, or be in a cell referenced by an address or range name. A logical formula is one that contains one of these logical operators:

=	Equal to
>	Greater than
<	Less than

- >= Greater than or equal to
- <= Less than or equal to
- <> Not equal to

- Error values include #DIV/0!, #N/A, #NAME?, #NULL!, #NUM!, #REF!, and #VALUE!. In a function, error values may be entered directly, result from a formula, or be contained in a cell referenced by an address or range name. A brief meaning of each of the error values is given here:

#DIV/0!	You tried to divide by zero
#N/A	Not available
#NAME?	Excel does not recognize a name
#NULL!	Two ranges you expected to intersect do not
#NUM!	Excel has a problem with a number
#REF!	Excel cannot find a cell or range reference
#VALUE!	You used the wrong type of operand or argument

Entering Functions

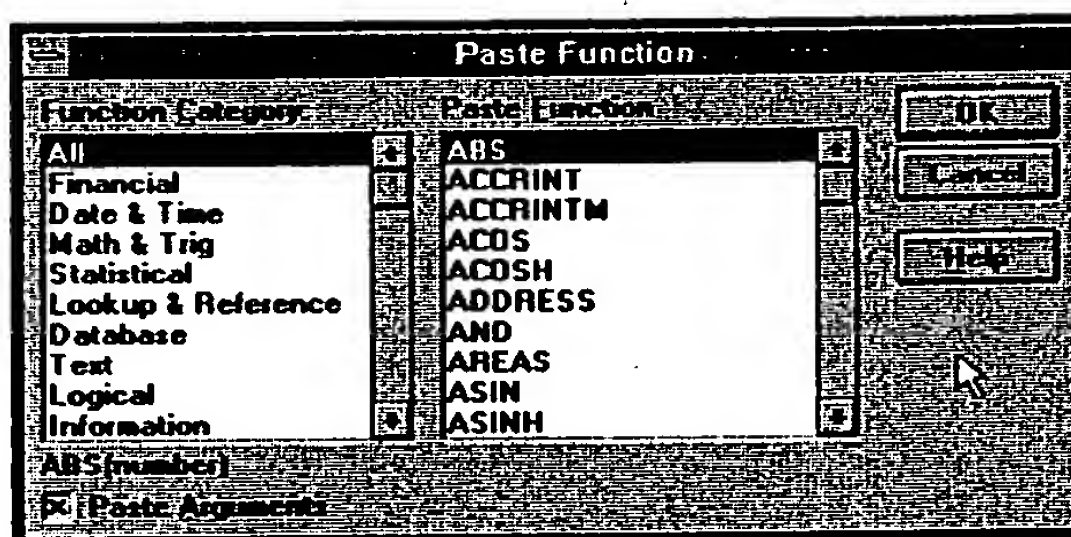
There are many different functions, but they all have the same structure, or syntax. A *syntax* is a set of rules for consistently doing something in an orderly manner—in this case, entering functions. The syntax for entering functions is as follows:

- Every function begins with the = symbol, unless it is inside a formula (that is, not the first element of the formula) or another function.
- Functions can be entered in either upper- or lowercase letters. They are displayed in uppercase by Excel. If you type a function in lowercase letters and Excel does not change it to uppercase, you know that you misspelled the function name or made some other mistake.
- Spaces cannot occur anywhere in a function, except within a literal string enclosed in quotation marks or immediately after a comma between arguments.
- The arguments of a function must be enclosed in parentheses. If one or more functions are used as arguments for other functions,

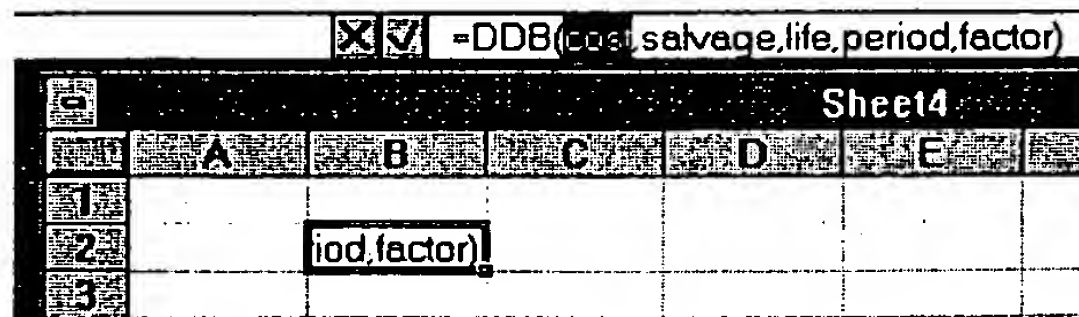
the parentheses must be nested, with complete left and right sets of parentheses for each function. Even functions that do not have arguments must have a set of parentheses. For example, NOW().

- Two or more arguments within a function are separated by commas. You should not have more commas than there are arguments or two arguments without a comma between them.
- Blank cells referenced in a function are assigned the value 0.
- Functions can be used by themselves as a formula or as a part of another formula, function, or macro function.

Functions may be directly entered by typing them in a cell, following the syntax just described, or you can have Excel build the formula using the Paste Function option on the Formula menu. To do the latter, make the cell in which you want the function the active cell, and then choose Paste Function from the Formula menu. The Paste Function dialog box opens, as shown here:



In the Paste Function dialog box, functions are listed alphabetically, within 12 categories including All. You can use the scroll bar to select a function, or you can type the first letter of a function to quickly jump closer to it. Click on the Paste Arguments option box, and Excel provides placeholders for the arguments to remind you what they are. If you choose to paste arguments, you must replace the placeholders with actual arguments. An example of the DDB (double declining balance) function with pasted arguments is shown here:



Additional Functions

The next several sections of this chapter discuss some of the functions that have not been discussed elsewhere. They fall into five groups: financial, informational, logical, mathematical, and text functions. Due to their number, not all of the individual functions are covered here. The following sections contain tips and suggestions for each of the five groups, along with one or two examples of functions within each group.

Financial Functions

Financial functions such as the following calculate amounts used in financing, budgeting, and depreciation. Optional arguments to the functions are in square brackets.

FV(*interest*,*term*,
payments[,*pv*,*type*])

Returns the future value, given a series of equal payments, the interest rate, and the term. Optionally you can enter the present value and/or whether the payment is made at the end of the period (the default, *type* = 0) or at the beginning of the period (*type* = 1).

IRR(*range*[,*guess*])

Returns the internal rate of return for a series of cash flows contained in the range. A guess may speed up the calculation. If you don't enter a guess, Excel uses 10%.

NPV(*interest*,*range*)

Returns the net present value of a series of cash flows contained in the range at a given interest rate.

**PMT(*interest, term,*
principal[, fv, type])**

Returns the payment required for a loan amount (principal) given the interest rate and loan term. Optionally you can enter the future value and/or whether the payment is made at the end of the period (the default, *type* = 0) or at the beginning of the period (*type* = 1).

**PV(*interest, term,*
payment[, fv, type])**

Returns the present value, given a series of equal payments, the term, and the interest rate. Optionally you can enter the future value and/or whether the payment is made at the end of the period (the default, *type* = 0) or at the beginning of the period (*type* = 1).

SLN(*cost, salvage, life*)

Calculates depreciation expense for an asset using the straight-line depreciation method.

When you use financial functions, the term and interest rate must be in the same time units. For example, if you want a result in months, the term must be in months and the interest rate must be in months. (See the first example in the “Examples” section that follows.)

The interest rate can be entered in a financial function as either a decimal (.108) or a percent (10.8%). Also, in many financial functions you must distinguish between cash outflow, which should be a negative number, and cash inflow, which should be a positive number. (See the examples that follow.)

Where a series of payments is used in a function, the payments are assumed to be equal, at regular intervals, and at the end of each period. This is known as an *ordinary annuity*. If you want to change the payment to the beginning of the period, use the optional *type* argument with a value of 1.

Examples To calculate the prospective monthly mortgage payment on a \$140,000 30-year loan at 10.8% interest, use the following function:

=PMT(.108/12, 30*12, 140000) = -\$1,312.14/month

To calculate the annual rate of interest necessary for a \$10,000 investment to grow to \$24,000 over 10 years with monthly compounding, use the following function:

`=RATE(10*12,0,-10000,24000)*12=8.79%/year`

Informational Functions

Informational functions such as the following provide information about cells and areas of the worksheet, including the number of rows or columns in a range, the formatting of a cell, and whether a cell is blank, contains text, or is a logical value.

<code>COLUMNS(range)</code>	Returns the number of columns in a range
<code>ISBLANK(value)</code>	Returns the logical value True if the value or cell is blank
<code>ISTEXT(value)</code>	Returns the logical value True if the value or cell is text
<code>ROW(reference)</code>	Returns the row number (not the number of rows) of the first row in the reference or an array of row numbers for all of the rows in the reference

Example Often it is helpful to know the column width of a cell. You can choose Column Width from the Format menu, or you can use the function `CELL("width")`. `CELL("width")` returns a value that is rounded to the nearest whole number. For example, `=CELL("width")` returns 8 for the standard cell width of 8.43.

Logical Functions

Logical functions such as the following perform tests to determine if a condition is true.

<code>AND(condition 1, condition 2,...)</code>	Returns True if all conditions or logical statements are true
<code>IF(condition,true-result,false-result)</code>	Evaluates an equation or condition for true or false and takes one action for a true result and another action for a false result
<code>TRUE</code>	Returns a logical True

Example When you calculate percentages, there are situations that result in dividing by 0 and produce a #DIV/0! error. To replace a possible error value with 0 in the formula =E15/C15, use the following formula in its place:

=IF(ISERR(E15/C15),0,E15/C15)

Mathematical Functions

Mathematical functions such as the following calculate general, matrix, and trigonometric values.

ABS(<i>x</i>)	Returns the absolute value of a number
ATAN(<i>x</i>)	Returns the arctangent of a number
MINVERSE(<i>array</i>)	Returns the inverse of a matrix or array
RAND()	Returns a random number between 0 and 1
ROUND(<i>x</i> , <i>n</i>)	Rounds a number off to a specific number of decimal places
SIN(<i>x</i>)	Returns the sine of an angle
SQRT(<i>x</i>)	Calculates the square root of a number

Angles used as arguments for COS, SIN, and TAN must be expressed in radians. To convert degrees to radians, multiply the degrees by $\text{PI}/180$. The angle that results from ACOS, ASIN, ATAN, and ATAN2 is in radians. To convert radians to degrees, multiply the radians by $180/\text{PI}$.

Example To calculate the length of a guy wire that is supporting a 150-foot-high antenna when the guy wire, attached to the top, makes a 55-degree angle with the ground, use the following function:

=150/SIN(55*PI/180) equals 183.12 feet

This function returns an answer of 183.12 feet.

Text Functions

Text functions convert, parse, and manipulate text strings. Some text functions are as follows:

CHAR(<i>x</i>)	Returns the ASCII character corresponding to the number <i>x</i>
------------------	--

EXACT(<i>string1</i> , <i>string2</i>)	Compares two text strings and returns True if the two strings are the same and False if they differ
LEN(<i>string</i>)	Returns the number of characters in a text string
MID(<i>string</i> , <i>start-number</i> , <i>n</i>)	Returns the specified number of characters from within a text string beginning at a specified position
PROPER(<i>string</i>)	Converts the first character in each word of a text string to uppercase and the rest of the characters to lowercase, as in a proper name
TEXT(<i>x</i> , <i>format</i>)	Converts a number to text with a given numeric format

The offset number used in string functions always begins at 1. The first character in a string is 1, and the last character is the length of the string. Blank cells in a string function are still considered text, have a length of 0, and do not return an error code.

Example To convert the date 4/15/92 in A1 to a text string that can be used in a title, use the following function:

```
=TEXT(A1, "mmm d, yyyy")
```

This function returns April 15, 1992, which is text, not a value.

Macros

A macro is a shortcut. It is a way of accomplishing a set of Excel commands with fewer steps and a way to automate or speed up repetitive procedures. A macro is also a way to guide a less knowledgeable user through a complicated worksheet.

There are two kinds of macros in Excel. A *command macro* is a series of Excel commands, and a *function macro* is a custom function that returns a result. An example of a command macro is one that saves your worksheet, while a function macro example is one that calculates your local sales tax. You can have Excel execute a command macro by pressing two keys. A function macro is executed by putting it in a worksheet cell and recalculating the

worksheet. Almost all commands that you can perform from the keyboard, the mouse, or a menu can be stored in a macro and can be activated as you choose. In addition to keyboard and menu commands, a set of *macro functions* lets you perform built-in programming functions, such as repeating a sequence or accepting input from the keyboard. With macro functions you can build custom menus and automate a worksheet. You can see that function macros, which are custom functions that return a result, and macro functions, which supply programming commands to Excel, are quite different.

Anything that you do on a repetitive basis is a candidate for a macro. Macros are stored on a separate sheet called a macro sheet. You can create a library of macros that you can use with many worksheets, which makes macros you create even more useful.

Macro Basics

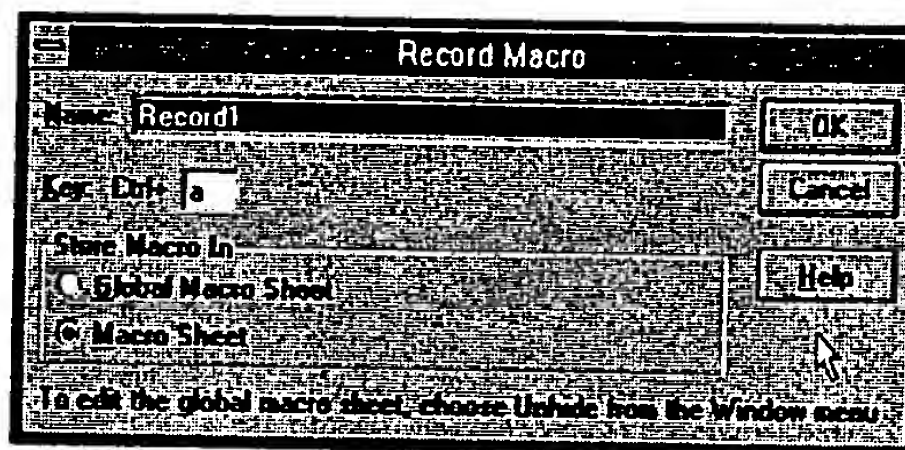
Few Excel tasks are more repetitive than saving a file. If you take normal precautions, you save your current worksheet several times each hour. To save an existing file, you either choose Save from the File menu, click on the Save tool from the Standard toolbar, or press **(SHIFT)-(F12)** or **(ALT)-(SHIFT)-(F2)**. Depending on whether you are using a mouse or the keyboard, this takes a varying number of keystrokes or mouse moves—not many, and ones with which you are probably familiar. When you repeat these actions 20 times a day, however, they begin to add up. If you could replace the actions with two keystrokes familiar to you, say **(CTRL)-(s)**, it might encourage you to save your files more often. Saving an existing file, then, is a good candidate for a macro.

Recording a Macro

Built into Excel is the capability to record whatever you are doing on an Excel worksheet and storing those steps on a macro sheet. Once stored, you can “play back” the steps and repeat what you were doing. The steps that are stored on the macro sheet comprise the macro, and playing them back is called running the macro. You turn on the Excel macro recorder by choosing Record from the Macro menu. Do that now and record a Save macro with these instructions:

1. Choose New from the File menu and click on OK to open a new worksheet.

2. Choose Save from the File menu, type `c:\sheet\macro`, and press **(ENTER)**. Since you want to build a macro to save a worksheet that has already been saved, you must start with a worksheet that has been saved.
3. From the Macro menu choose Record. The Record Macro dialog box opens, as shown here:



There are two ways you can store macros: on a global macro sheet, or on a macro sheet you create. The global macro sheet is automatically opened each time you start Excel. This is generally the macro sheet to use for simple, frequently used utility macros because Excel will save, name, and open the sheet for you. If you create a new macro sheet, you will have to manually open the sheet both to record to it and to run the macros once they are recorded. You can record a macro by selecting the Macro menu and choosing Record, or by displaying the Macro toolbar and clicking on the Record Macro tool. You will create and record a macro using the global macro sheet in this section.

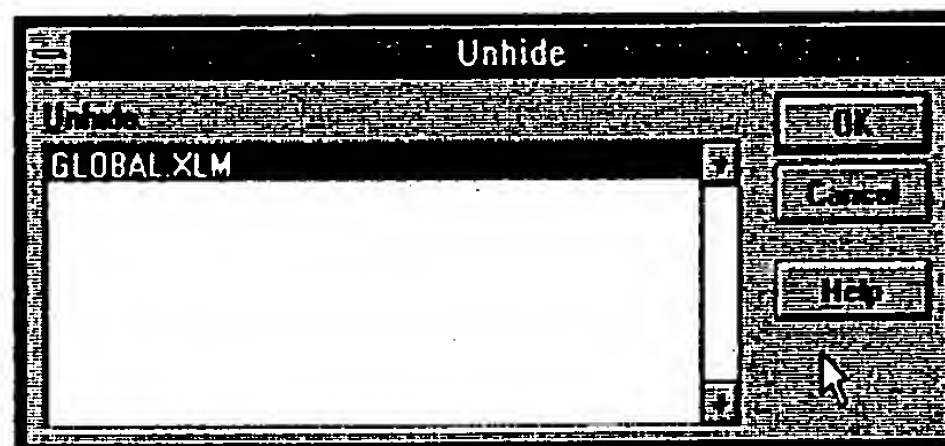
4. Type **Save.Worksheet** as the macro name, press **(TAB)** to move to the Key field, and type **s**.
5. Click on Global Macro Sheet in the Store Macro In list box and click on OK. The dialog box closes, and the Recording status message comes on in the Status bar.

The name of a macro can be any legitimate Excel name. It must start with a letter, can be up to 255 characters long, and can contain any combination of letters, numbers, periods, and underlines. It should not look like a reference (either D3 or R3C4) and cannot contain spaces. Since you cannot

use spaces, periods or underlines are used as word separators. Periods are used in this book. An Excel name can be entered in either upper- or lowercase letters—Excel does not distinguish between the two.

The shortcut key can be any single upper- or lowercase letter. Upper- and lowercase letters are considered two different characters and will not conflict with one another. You cannot use numbers as shortcut characters.

6. Choose Save from the File menu. This is the step you want to record.
7. Choose Stop Recorder from the Macro menu. The Recording message disappears.
8. From the Window menu, choose Unhide. The Unhide dialog box opens as shown here:

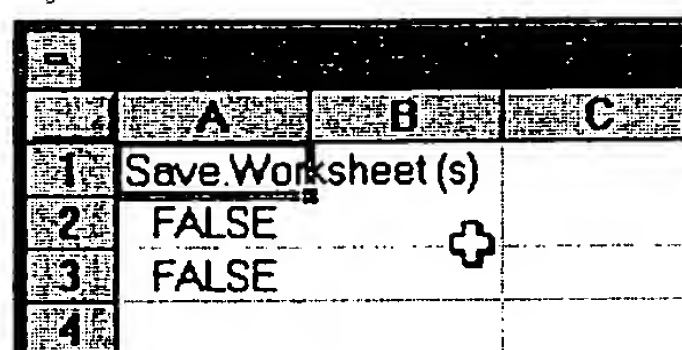


9. Double-click on GLOBAL.XLM. The global macro sheet that was automatically created in the previous steps becomes the active sheet, as shown here:

GLOBAL.XLM	
A	B
1	Save Worksheet(s)
2	=SAVE()
3	=RETURN()
4	

The global macro sheet looks just like a normal worksheet except that the columns look a little wider. The macro itself is in the upper-left corner, in cells A1:A3. A1 contains the name and shortcut key you gave the macro. A2 and A3 are the macro functions that save the current worksheet and return control to you. Macro functions are formulas—they always begin with an equal

sign (=). A macro sheet always displays formulas, not the results they produce. Displaying formulas is an option on a normal worksheet, but normally a worksheet displays the results a formula produces. You can use the Options Display option to turn off the formulas display on a macro sheet, but the resulting values are generally not informative. Here is what your global macro sheet looks like with formulas turned off:



	A	B	C
1	Save Worksheet(s)		
2	FALSE	+	
3	FALSE		
4			

When you are displaying formulas instead of their results, everything on the worksheet is left aligned and you cannot change it with the Format alignment option. Otherwise, all formatting works on a macro sheet as it does on a worksheet.

Documenting a Macro

As you create macros, you may find that after a while you forget what they do. Also, you may want to give one or more macros to someone else to use, and they must know what the macros do. For this reason you must document your macros when you create them.

You can document a macro in several ways. You have used two forms of documentation already—giving the macro a descriptive name and using an obvious shortcut key. Other ways include formatting the macro name on the global macro sheet so it stands out, adding one or more cell notes, and, most importantly, adding some comments beside the macro commands. Add some comments and format the macro name with these steps:

1. With A1 as the active cell, click on the Bold button in the Standard toolbar, and click on OK. The macro name should be made bold.
2. Click on B1, type **Saves current wksht**, and press **(ENTER)**. After slightly widening column A, the upper-left corner of your macro sheet now looks like this:

GLOBAL.XLM	
A	B
1 Save Worksheet (s)	Saves current wksht
2 =SAVE()	+
3 =RETURN()	
4	

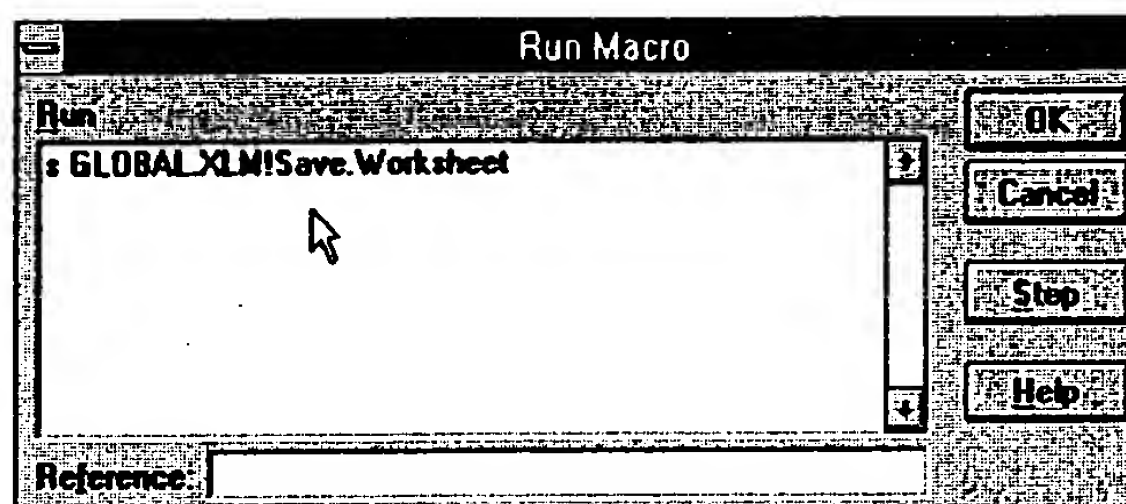
Running a Macro

Now that you have a finished and documented macro, you can run it in one of two ways. First and most simply, you can press **(CTRL)-(S)**, the shortcut key. Second, the Run Macro dialog box, reached by choosing Run from the Macro menu, lists all the macros available on open macro sheets, so you can select the macro you want and click on OK. Try both of these methods using the following instructions:

1. From the Window menu, choose MACRO.XLS.
2. Press **(CTRL)-(S)**. The file is saved.

If you look at the Reference area of the Formula bar or at your disk light, you will see a brief indication that the file was saved. Also, you may see the hourglass wait indicator come on briefly. Press **(CTRL)-(S)** several times until you are satisfied it is working.

3. From the Macro menu, choose Run. The Run Macro dialog box opens, as shown here:



The Run Macro dialog box lists the macro you have just created. On the left is the shortcut key followed by the name of the macro sheet and the macro

name. By clicking on the entry in the list box and then on OK you can run the macro.

4. Click on the entry in the list box and on OK (or double-click on the entry). Again you'll notice a brief flicker in the Reference area and in your disk light telling you the worksheet is being saved.

The Run Macro dialog box also serves as a reference if you should forget what the shortcut key is on a particular macro.

If you had any fears about macros, you can now set them aside. You have successfully created and run a macro!

5. Activate the global macro sheet by choosing it from the Window menu.
6. Close the global macro sheet by choosing Hide from the Window menu. When you exit Excel, a dialog box will ask if you want to save changes to the global macro sheet. Click on Yes and the global macro sheet will be hidden, but working, in future uses of Excel.

To edit the global macro sheet, choose Unhide from the Window menu. The Unhide dialog box appears. Highlight GLOBAL.XLM, click on OK, and the file is opened.

If you exit Excel without both hiding the global macro sheet again and choosing Yes to save the change to it, the global macro sheet will appear every time you start Excel until you choose to hide it and to save the change on exiting.

Remember that File Save erases the current file on disk before replacing it with the file being saved. *This means data can be lost.* You may want to create a backup file through the Options command button on the File Save As dialog box. This gives you the added protection of preserving the last file saved.

Repeat the steps you went through to record the Save.Worksheet macro on the global macro sheet, but this time store the macro in a new macro sheet. As a review:

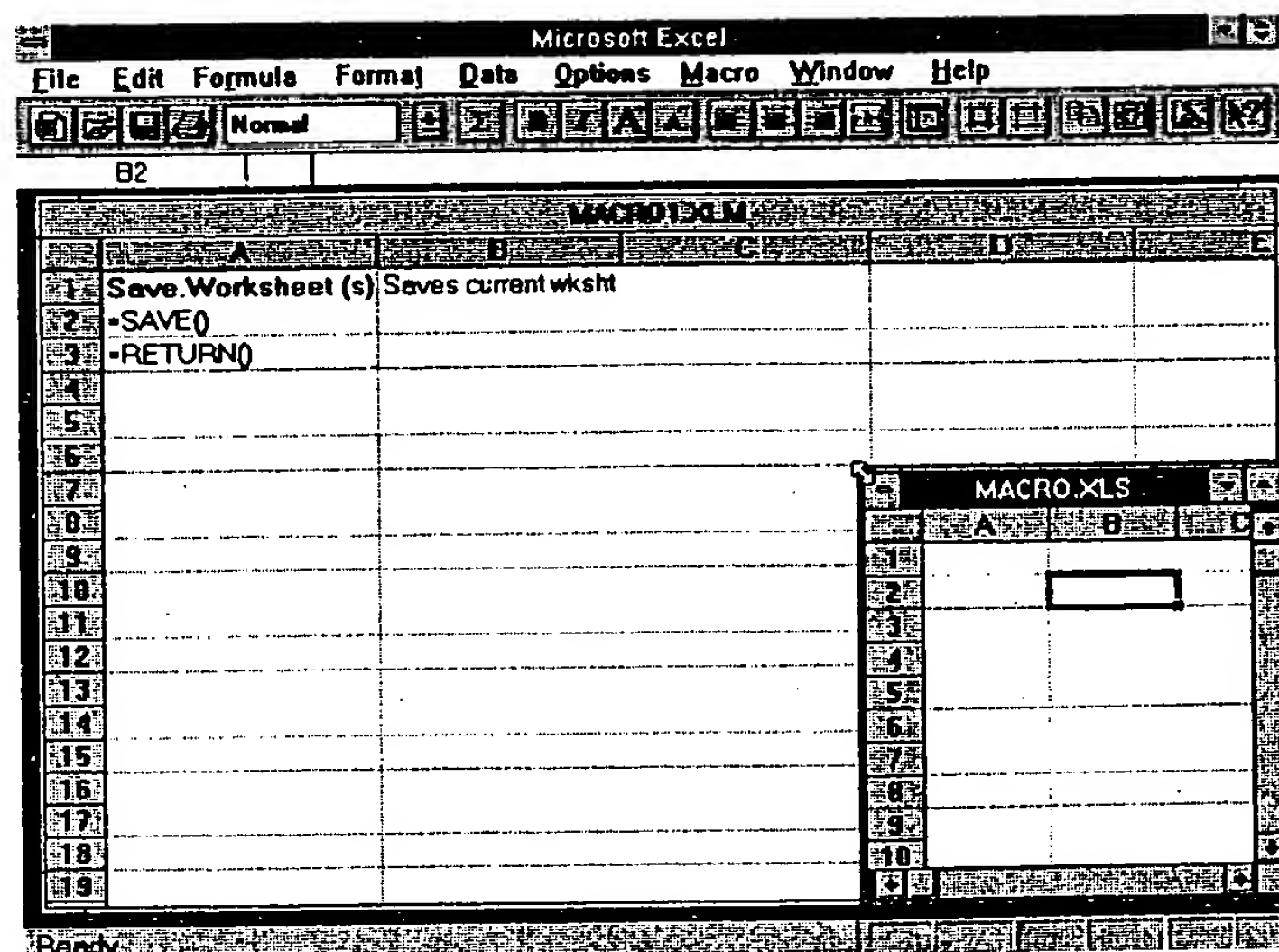
1. Choose Record from the Macro menu.

2. Reenter the macro name and key (Save.Worksheet and s). Choose Macro Sheet and click on OK.
3. Choose Save from the File menu to record it and then choose Stop Recording from the Macro menu.
4. Click on Macro1 in the Window menu and make cell A1 bold.
5. Click on B1 and repeat the documenting of the Save macro.
6. Open MACRO.XLS from the Window menu.

Creating Additional Macros

Create several more simple and general-purpose macros. This time, however, watch the macros being built by reducing the size of the worksheet you are working on and exposing most of the macro sheet, as shown in Figure 10-8. Drag the upper-left corner of the MACRO.XLS worksheet window to reduce it to approximately the size shown in Figure 10-8. When you create a second macro, Excel places it at the top of the next available column of the current macro sheet, unless you tell Excel otherwise with the Set Recorder

Figure 10-8. A small worksheet set up for watching macros being created



option in the Macro menu. Since you have used columns A and B, the next macro you create is placed in C1.

Copy Macro

Another heavily used option is Copy. While it has a well-defined shortcut key built into Excel, many people find it hard to remember. Create a macro for Copy and assign it the intuitive shortcut key **CTRL-C**.

1. From the Macro menu, choose Record, type **Copy.Selection**, press **TAB**, type **c**, and click on OK. The Recording message comes on in the Status bar, and the name and shortcut key appear in C1 on the macro sheet.

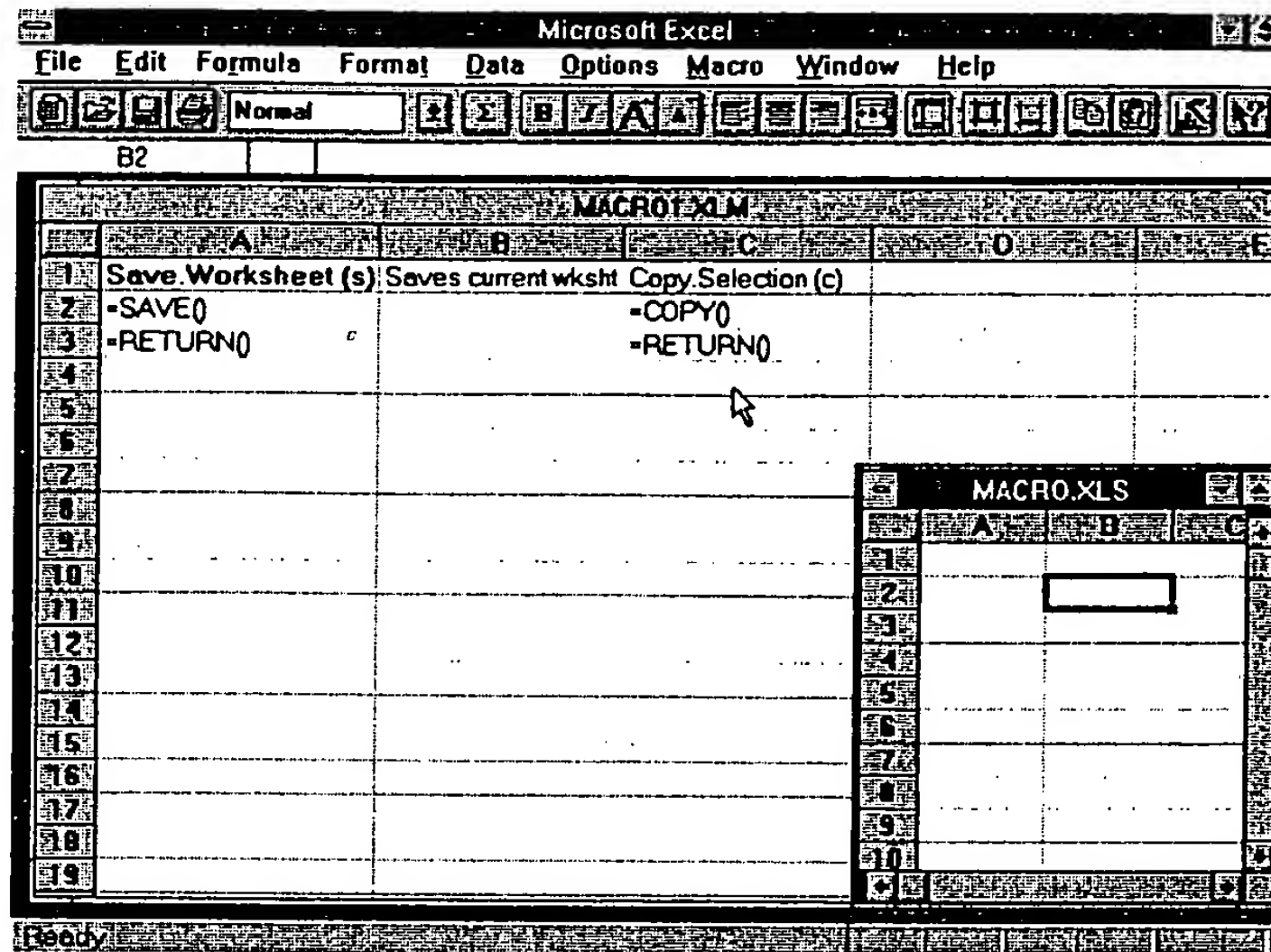
If you cannot see the **Copy.Selection** macro name on your macro sheet, it is because you left Excel between creating the Save macro and this macro. If you start a new session, even if you open your old macro sheet, Excel creates a new macro sheet to use for macros created in the current session, unless you tell it otherwise. (You will see how in a moment. For now, carry on creating the Copy macro even though you cannot see it.) In step 4, choose the new macro sheet from the Window menu instead of the old one.

2. Choose Copy from the Edit menu. The blinking marquee appears around B2 (or whatever cell you are pointing at) in **MACRO.XLS**. The macro function **=COPY()** appears in C2 on the macro sheet.
3. Choose Stop Recorder from the Macro menu. The macro function **=RETURN()** appears in C3 on the macro sheet. Press **ENTER**. Your screen should look like that shown in Figure 10-9.
4. Click on the macro sheet and on cell C1. Click on the Bold tool in the Standard toolbar.
5. Click on D1, type **Copies current select**, and press **ENTER**. Your second macro is documented.

Setting a Recording Range

Your next macro would be placed in E1, unless you tell Excel otherwise. Since that is off the screen, tell Excel you want it to begin in A5. You do that either by selecting a starting cell, in which you want the macro to start or by

Figure 10-9. A copy macro added to macro sheet



selecting a range you want the macro to occupy and then choosing the Set Recorder option from the Macro menu.

If you select a single cell in which to start the macro, Excel fills as many cells below that cell in the same column as necessary to complete the macro. If the macro reaches the bottom of the column, Excel redirects the macro to the top of the next column with a GOTO macro function, and then continues the macro in the next column. If you select a single cell and the single cell is not blank, Excel finds the last nonblank cell in the column and begins recording immediately below it. If the last nonblank cell has the RETURN macro function in it, RETURN is replaced by the first macro function of the new macro. In this way you can stop recording a macro and then later restart where you left off.

If you select a range in which to record a macro, the range becomes the limits within which the macro is contained. Excel starts the macro in the upper-left corner and continues to the lower-right corner, placing GOTO functions at the bottom of each column. If Excel reaches the limits of the range without completing the macro, you get a message that the range is full.

Unlike selecting a single cell, if you select a range and the first cell in the range is not blank, Excel displays a message saying the range is full and cannot be used.

Set the starting cell for recording the next macro with these instructions:

1. Click on A5 of the macro sheet.
2. Choose Set Recorder from the Macro menu.
3. Choose MACRO.XLS from the Window menu to return to your worksheet.

Once you have set where you want to place your next macro, you can start the macro with either the Record or Start Recorder option on the Macro menu. Both options start in the cell you set. The principal difference is that Record is meant to start a new macro and Start Recorder is meant to be used for adding to the last macro you have entered. Start Recorder does not ask you to name or enter a shortcut key for a macro. You can use it only if you have already used Record within an Excel session (since you have most recently started Excel) or if you have used Set Recorder to establish a starting cell. If you have started a macro with Record, you can use Stop Recorder to quit macro recording and then restart with Start Recorder as if you had never quit; you need not use Set Recorder.

Formatting Macro

The custom percent format 0.0% deserves a macro for quick use. Build it next and give it the shortcut keys **CTRL-P**.

1. Choose Record from the Macro menu, type **Percent.Format**, press **TAB**, type **p**, and click on OK. The name and shortcut key appear in A5 of the macro sheet.
2. Choose Number from the Format menu, click on Percentage, select 0.00%, delete one decimal zero in the text box, and click on OK.
3. Choose Stop Recorder from the Macro menu to complete the macro.
4. Click on the macro sheet, click on A5, and make the name bold using the Bold tool. Then in B5 type **0.0% Format**, and press **ENTER**.
5. Widen column A by dragging on the intersection between columns A and B so you can see the full format macro function, as shown in Figure 10-10.

Date- and Time-Stamp Macro

You will often want to add the date and/or time to a worksheet. Manually you must type `=NOW()`, format the cell, and then use Copy and Paste Special to convert the function to a permanent value that does not change every time the worksheet is recalculated. It is easier to look at your watch or calendar and type the numbers as text so they do not have to be formatted. A macro takes this process down to two keystrokes that format the cell, enter the function, and convert it to a value. Create the macro following these steps:

1. Click on A9 of the macro sheet and choose Set Recorder.
2. Choose MACRO.XLS from the Window menu and click on A1 if the active cell is not already there.
3. Choose Record, type **Date.Time.Stamp**, press **(TAB)**, type **d**, and click on OK. The name appears in A9.
4. Type `=now()` and press **(ENTER)**.
5. Choose Number from the Format menu, select the m/d/yy h:mm format, and click on OK.
6. Widen column A of the worksheet by dragging on the intersection between columns A and B until column A is about half again as large.
7. Choose Copy from the Edit menu, choose Paste Special from the Edit menu, click on Values and OK, and press **(ESC)**. The `=NOW()` formula is converted to a value and the copy marquee removed.
8. Choose Stop Recorder from the Macro menu. Click on the macro sheet, make the macro name bold, and document your macro as shown in Figure 10-11.

Now try out this macro.

9. Choose MACRO.XLS from the Window menu, click on B3, and press **(CTRL)-(d)**.

Figure 10-10. A macro sheet with the Percent macro

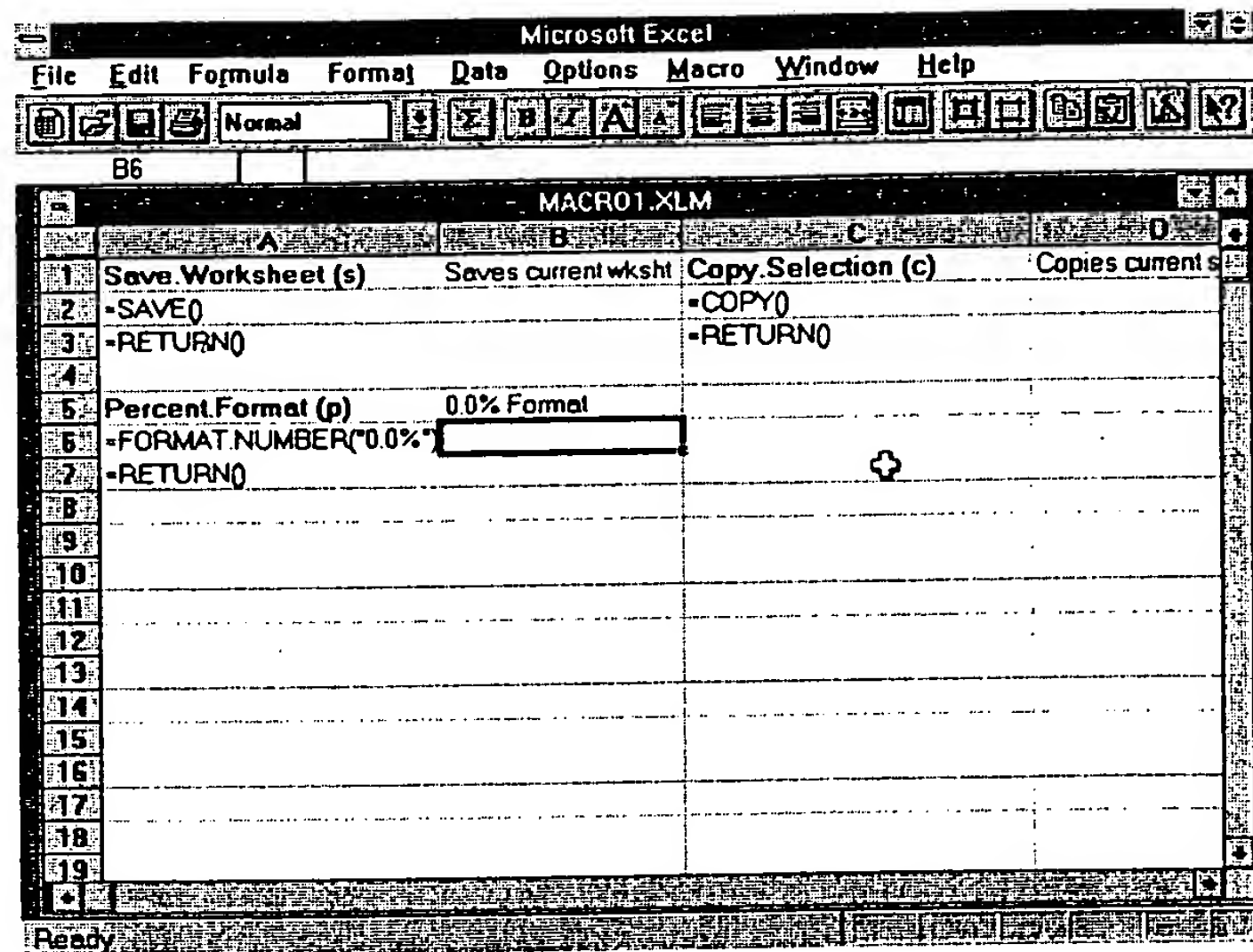
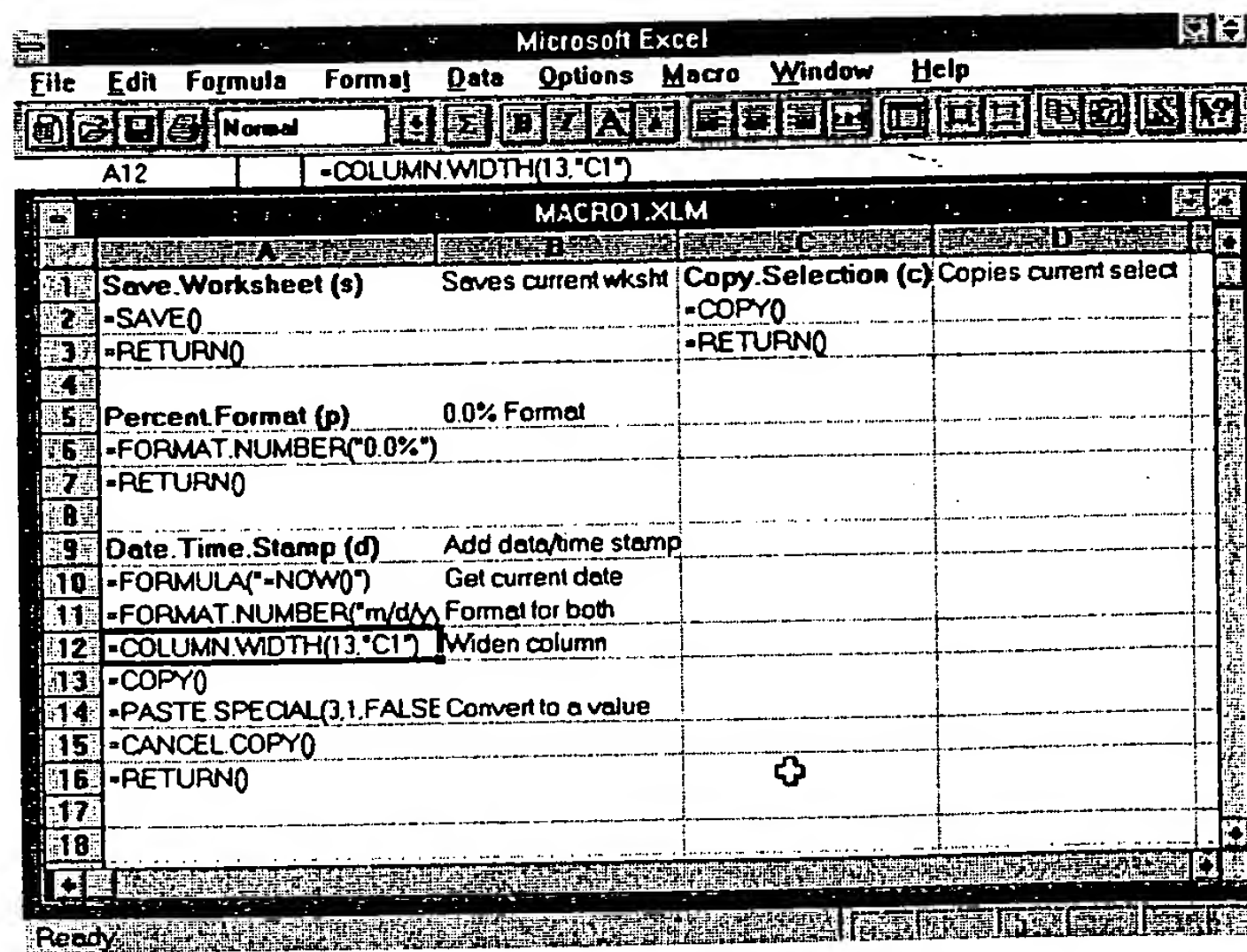


Figure 10-11. Date/Time Stamp macro



The date and time are placed in B3 and it is formatted (although you cannot tell), but the column is not widened. Go back to the macro sheet and see why.

10. Click on the macro sheet and widen column A, if necessary, until you can see all of the COLUMN.WIDTH macro function, as shown here:

9	Date.Time.Stamp (d)	Add data/time stamp
10	=FORMULA("=NOW()")	Get current date
11	=FORMAT.NUMBER("m/d/yy")	Format for both
12	=COLUMN.WIDTH(13,"C1")	Widen column
13	=COPY()	

The COLUMN.WIDTH macro function has two arguments: the width itself (your width may be different due to variation in dragging), and an optional reference. The reference shown here, C1, does not mean column C, row 1, but rather "column 1." This is the problem. To make the Date/Time Stamp macro flexible, you need to remove the column reference. Then the COLUMN.WIDTH macro function refers to the current selection, which is what you want. If your width value is a number with many decimal places, edit it also to round it to an even 12.

11. Click on A12. Edit A12 so that it contains =COLUMN.WIDTH(12).
12. Return to the MACRO.XLS worksheet and try the macro again in B5. It now works the way it should.

Entering Macros

All the macros you have created so far have been recorded. You modified the last recorded macro to give you what you want. You can also directly type in a macro. You can use either an existing macro sheet or create a new one. You simply pick an area on the macro sheet and start typing the necessary macro functions. This, of course, takes some familiarity with the macro functions and their arguments. Once you have written the macro, you can name it with the Define Name option on the Formula menu. Try that next by

writing a macro to apply the #,##0 format. You can use the Percent macro as a model.

1. Click on the macro sheet and on C5. Type **Comma.Format(f)** and press **(DOWN ARROW)**.
2. Type **=format.number("#,##0")**, press **(DOWN ARROW)**, type **=return()**, and press **(UP ARROW)** twice.
3. Choose Define Name from the Formula menu, click on Command as the macro type, press **(TAB)**, type **f** as the shortcut key, and click on OK. (The name Comma.Format is already in the Name text box.)
4. Format the name on the macro sheet to make it bold, and enter the documentation as shown here:

Comma.Format(f)	#,##0 Format
=FORMAT.NUMBER	
=RETURN()	

5. Try out the macro by returning to the MACRO.XLS worksheet, typing **62503.60**, pressing **(ENTER)**, and pressing **(CTRL)-(f)**. It works! Next press **(CTRL)-(p)**, since you have not tried that macro, and format the number as a percent. It also works.

You can see that while you can type in a macro, it is much easier to use the recorder. Even if you need to do some heavy modification, building an initial structure with the recorder is a substantial benefit. It not only saves time, but also it gives you the correct macro function name and argument set. It saves you from either having to look these up or use the Paste Function option of the Formula menu.

Rules for Macro Entry

All parts of a macro can be typed in either upper- or lowercase. Excel converts it to uppercase if it is spelled correctly. A macro can occupy as many cells in a column as necessary. You can use both worksheet functions and macro functions. Place one function per cell. This makes the macros easier

to read on the screen and easier to edit. As Excel is executing a macro and completes the instructions in one cell, it automatically goes to the cell immediately below and continues macro execution. Excel continues down a column in this manner until it reaches a terminating or redirecting macro function such as RETURN, GOTO, or HALT. During macro execution, Excel ignores a blank cell.

If a macro refers to a range, it is better to name the range than to use addresses. An address in a macro is not updated if the worksheet is changed. A range name, on the other hand, continues to track an address with changes in the worksheet. Also, if you are working with multiple files, it is a good idea to precede a range name with the filename followed by an exclamation point.

Debugging Macros

When a macro does not behave the way you expect, you want to *debug* it, or correct it. Debugging can be as simple as correcting an obvious spelling error in a range name. In many instances, however, the error is not so obvious, as with the COLUMN.NUMBER problem in the Date/Time Stamp macro.

Debugging begins by looking carefully at what happened when the macro was run. Were there any error messages, and what did they say? What happened on the worksheet? Do you get the same result each time you run the macro?

Next, look at the macro itself. Are there any misspellings? Are there missing arguments, periods, or macro functions? If you recorded the macro, did the recorder supply some arguments you do not want? Did you use absolute addressing when you wanted relative or vice versa? Have you defined the range and macro names you are using?

Finally, go back and record the macro again to see if you get the same macro functions a second time. Carefully note all of your actions while you are recording the macro.

In most instances these steps identify the problem. If not, there is one further thing that can be done. You can use the Step command in the Run dialog box to execute a macro one step (one macro function) at a time. Excel will pause after each step so you can see the effects of that step, and continues only when you give the command to do so. The Step command also allows you to permanently halt the macro after any step, so you can more fully explore the partial results of a macro. Try the Step command now by choosing

the DATE.TIME.STAMP macro from the Run dialog box. Click on Step and the Single Step dialog box appears. Keep clicking on Step Into to see the macro execute one step at a time.

Macro Functions

A macro function, when executed, performs a predefined function that may be available from the keyboard, the menus, or the mouse. Some macro functions, however, are for purposes of further automating a process and are not available in any other form. These macro functions cause the macro to accept input from the user, wait while the user does something, make a choice among several things, or loop through a set of macro functions multiple times. These are programming macro functions, and Chapter 12 works with them extensively.

Macro functions have a common syntax that must be followed in order for Excel to understand what to do. This syntax is exactly the same as that for worksheet functions described earlier in this chapter.

Function Macros

Function macros are custom functions you create on a macro sheet and then use on a worksheet to return a value or other result. You can distinguish function macros from command macros in two ways. First, command macros take some action like formatting, copying, or saving. Function macros take no action, but rather produce a result, like you might get from a calculation. Second, a command macro is wholly contained on the macro sheet and is executed with either the shortcut key or the Run option on the Macro menu. A function macro is created on a macro sheet, but to use it you must enter the resulting function on a worksheet.

Function macros are generally calculations you have to perform over and over. They have arguments through which you supply values and they use formulas and regular functions to calculate a result based on the values you supply. When you build a function macro, you must use three special macro functions that handle the arguments and the result. These functions, in the order in which they must be used, are as follows:

RESULT

The RESULT function is used only if you need to change the data type of the result. The RESULT function has one argument, the data type number. If you have not used a RESULT function to change it, the result is assumed to be a number, text, or a logical value. The possible data type numbers are

1	Number
2	Text
4	Logical
8	Reference
16	Error
64	Array

Data type numbers can be added together except for the reference and array types. For example, the default of number, text, or logical is a type 7 (1+2+4).

ARGUMENT

You must have one ARGUMENT function for each argument in the function macro you are building, and the arguments must be in the order in which they are presented in the function macro. The ARGUMENT function can have up to three arguments: a name, a data type, and a reference. You must have either a name or a reference. Whichever you specify, the other is optional. The data type is always optional. The name must be a legitimate Excel name and becomes defined by the ARGUMENT function. It can then be used by the formulas and regular functions that follow. If you do not specify a data type, Excel assumes it to be a number, text, or a logical value. If the value received by the ARGUMENT function is not a default type and you have not used the data type argument to change that, you will get a #VALUE! error. The reference argument is a cell or range reference on the macro sheet where the value received by the ARGUMENT function is placed. If you use both a name and a reference, the reference is given the name and can be referred to by it.

RETURN

All function macros must end with the RETURN function. The RETURN function has one argument in a function macro—the cell on the macro sheet that contains the result.

The formulas and regular functions to be used in a function macro must be placed after the last ARGUMENT function and before the RETURN function.

Function macros must be directly entered—they cannot be recorded. Build an example to see how they work. The example, call it Tax, calculates sales tax. It has two arguments: the amount on which to calculate the tax and the tax rate. Enter the tax function macro on the open macro sheet in C9.

1. Click on the macro sheet and on C9. Type **Tax** and press **(ENTER)**. Format it as bold.
2. From the Formula menu, choose Define Name. Click on Function and on OK.
3. Click on cell C10. Then type `=argument("amount",1)`, press **(DOWN ARROW)**, and then type `=argument("rate",1)`. Press **(DOWN ARROW)** again.
4. Type `=amount*rate`, press **(DOWN ARROW)**, type `=return(c12)`, and press **(ENTER)**.
5. Click on MACRO.XLS in the Window menu, and click on A5. (Drag the MACRO.XLS window to the left if you want to see the macro while you are working on the worksheet.)
6. From the Formula menu, choose Paste Function, scroll the Function Category list box and click on User Defined. Then click on the name MACRO1!Tax, and click on OK. You are left with the function in the Edit area and the word "amount" highlighted.
7. Press **(DEL)** six times to remove the arguments, then type `100,8.1%` and press **(ENTER)**.

The result, 8.1, appears in A5, as shown in Figure 10-12. If an argument is missing when a function macro is used, the ARGUMENT function for that argument passes a value of #N/A to the formulas and functions that follow. To allow for optional arguments, you must trap the #N/A with an IF(ISNA())

Figure 10-12. Function macro for calculating sales tax

A5		=MACRO1.XLM!Tax(100,8.1%)		
MACRO1.XLM				
	A	B	C	D
1	Save Worksheet (s)	Saves current wksht	Copy Selection (c)	Copies current select
2	-SAVE()		-COPY()	
3	-RETURN()		-RETURN()	
4				
5	Percent Format (p)	0.0% Format	Comma.Format(f)	###0 Format
6	-FORMAT.NUMBER("0.0%")		-FORMAT.NUMBER("###0")	
7	-RETURN()		-RETURN()	
8				
9	Do	MACRO.XLS	Stamp	Tax
10	-FOR	A	Rate	-ARGUMENT("amount",1)
11	-FOR	B	Rate	-ARGUMENT("rate",1)
12	-C	1/24/92 21:19	Amount	-amount*rate
13	-C			-RETURN(C12)
14	-PA	8.1	Value	
15	-C	1/24/92 21:27	Argument	
16	-R	6250360.0%		
17				
18		1/24/92 22:02		
19				
20				

Ready

Figure 10-13. Tax function macro with an optional argument

A5	=MACRO1.XLM!Tax(100,8.1%)			
MACRO1.XLM				
	A	B	C	D
1	Save Worksheet (s)	Saves current wksht	Copy Selection (c)	Copies current se
2	-SAVE()		-COPY()	
3	-RETURN()		-RETURN()	
4				
5	Percent.Format (p)	0.0% Format	Comma.Format(f)	###0 Format
6	-FORMAT.NUMBER("0.0%")		-FORMAT.NUMBER("###0")	
7	-RETURN()		-RETURN()	
8				
9	Do	MACRO.XLS	Stamp	Tax
10	-FOR	A	Rate	-ARGUMENT("amount",1)
11	-FOR	B	Rate	-ARGUMENT("rate",1)
12	-C	1/24/92 21:19	Amount	-amount*IF(ISNA(rate),8.1%,re
13	-C			-RETURN(C12)
14	-P	8.1	Value	
15	-C	1/24/92 21:27	Argument	
16	-R	6250360.0%		
17				
18		1/24/92 22:02		
19				
20				

function. For example, if you want to make the tax rate optional in the function macro you just built, you must change the formula in C12 to `=amount*IF(ISNA(rate),8.1%,rate)`. This way, if the rate is not entered, 8.1% would be used, as shown in Figure 10-13.

As you are going back and forth between the macro sheet and the worksheet, be aware that the worksheet is not recalculated by simply activating it. You must either make an entry to or edit a cell, or press **(F9)**, the Calculate Now key.

Depending on who is going to use your macros, especially function macros, you must consider their error-handling capability. If you are the only one who will be using them, they can probably be fairly insensitive to errors. If novice Excel users are going to handle them, the function macros must work with many different error conditions. You are now in the position of a programmer whose hardest job probably is to figure out all the ways someone can use their program. A basic ground rule is that if it can happen, it will, and if it cannot happen, it might anyway. Chapter 12 deals entirely with writing macros to automate a worksheet so a novice can use it (and it does *not* consider every possible error that could occur!).

You are left with a number of open worksheets. Only the macro sheet has potential value. Save it if you want and exit both Excel and Windows by telling Excel you do not want to save the other worksheets.

CHULMAN'S UNDOCUMENTED CORNER

0193 MARCH 1993 ANIMAT PUBLICATION

Dr. Dobb's

JOURNAL

SOFTWARE
TOOLS FOR THE
PROFESSIONAL
PROGRAMMER

40

DATA STRUCTURES & FILE FORMATS INSIDE & OUT

++ TEMPLATES FOR
FILE CONVERSION

UNDOCUMENTED
WINDOWS

YOUR OWN
CD-ROM
REDIRECTOR

UNDOCUMENTED
BTVIEW

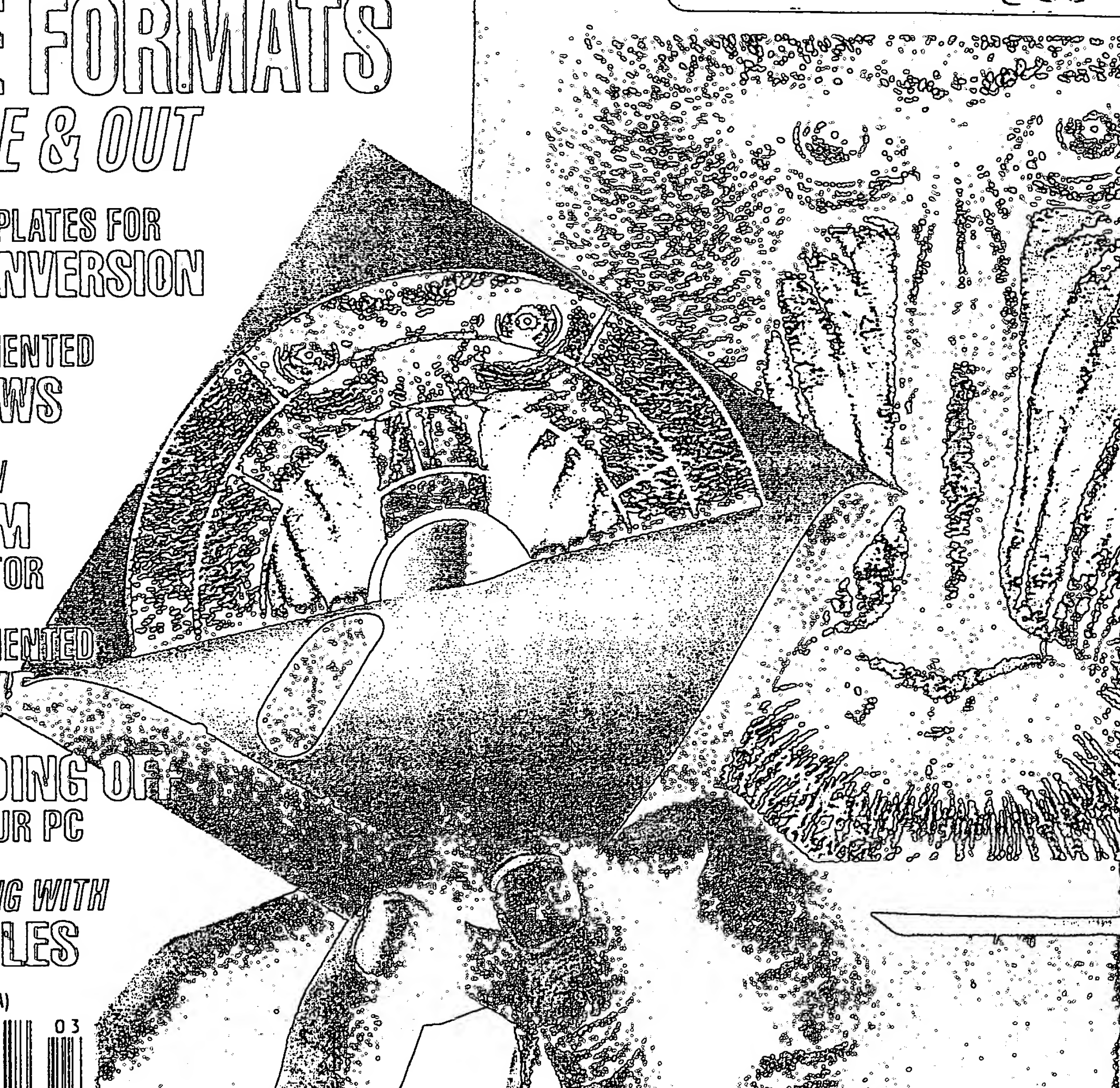
BOUNDING OFF
WITH YOUR PC

ANIMATING WITH
FLIC FILES

\$9.95 (\$4.95 CANADA)



03



FEATURES

THE FLIC FILE FORMAT

by Jim Kent

As their name suggests, flic files are a sequence of still frames which can be rapidly flipped through to achieve the illusion of movement—the software equivalent of movies. Among the applications and tools that support the flic file format are Autodesk's Animator, IBM's Ultimedia Tool Series, and Microsoft's Video for Windows.

18

FILE CONVERSION USING C++ TEMPLATES

by Timothy Butterfield

Tim shows how C++ templates can be used to build a parser-based "black box" conversion class that allows you to use various data types and processes without having to rewrite the basics of the conversion engine for each new combination.

26

COMPOUND DOCUMENTS

by Lowell Williams

Compound documents contain a mixture of different data—text, line art, raster graphics (images), tabular data, and even audio and video. Lowell makes the case that the now-familiar ASCII can no longer fulfill its role as a universal document-interchange standard and that we should begin examining compound ASCII alternatives like ODL, SGML, and CDA.

32

DESIGNING COMPLEX DATACENTRIC APPLICATIONS

by Paul Bonneau

Paul discusses the data structures and client/server architecture of HyperChem, a molecular modeling tool that runs on PCs and Silicon Graphics workstations. Implemented in about 500,000 lines of C code, HyperChem lets you can create three-dimensional atomic structures, visualize and manipulate their structural relationships, and perform classical and semi-empirical, quantum mechanical calculations.

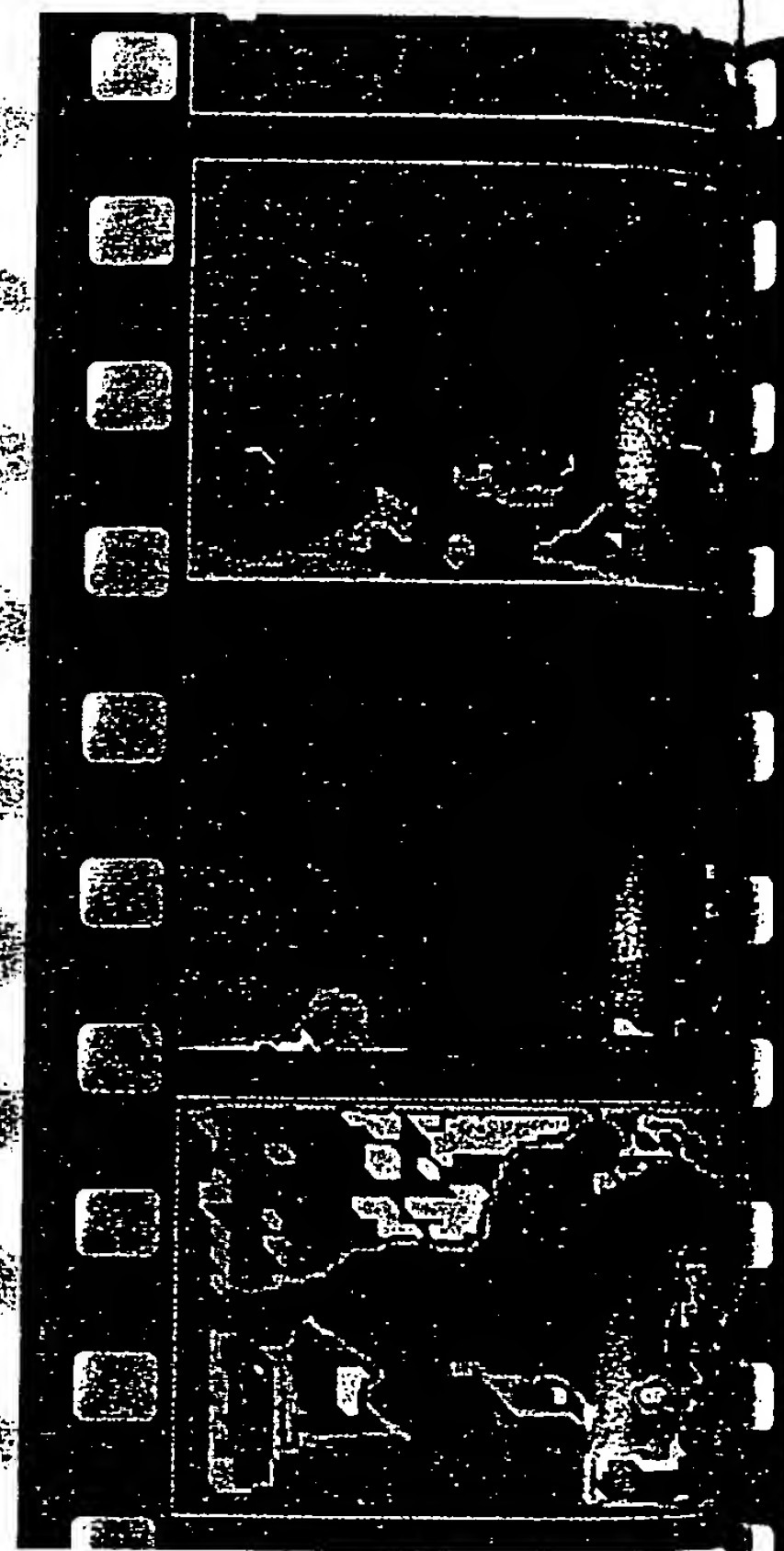
40

A DOS REDIRECTOR FOR SCSI CD-ROM

by Jim Harper

Getting data from CD-ROM to where your application can use it isn't straightforward under MS-DOS. Jim examines how this process works under DOS, then presents the code for an MSCDEX-like extension to MS-DOS that allows access to either High Sierra or ISO-9660 CD-ROMs. This redirector works in conjunction with a TSR-based driver for SCSI devices.

44



JULIAN AND GREGORIAN CALENDARS

by Peter J.G. Meyer

Peter presents a C function which converts any date within an 11-million-year period in either the Gregorian calendar or the Julian calendar into a unique number in the range of approximately -2,000,000,000 through 2,000,000,000.

52

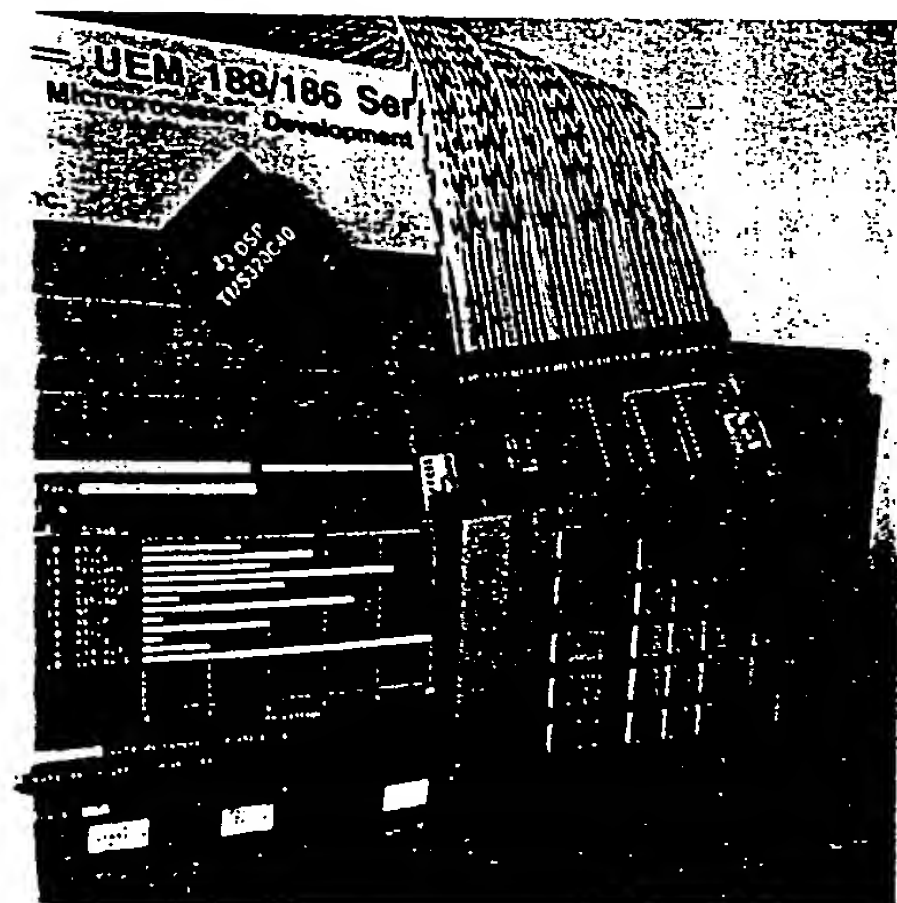
EMBEDDED SYSTEMS

TOOLS FOR EMBEDDED-SYSTEMS DEBUGGING

by Christopher Perez

Tools such as microprocessor in-circuit emulators, microprocessor on-chip debug circuitry, and logic analyzers can make the tough job of embedded-systems designers easier. Chris focuses on the JTAG specification and techniques for getting the most out of logic analyzers.

52



ENTS

MARCH 1993
VOLUME 18, ISSUE 3

NETWORKED SYSTEMS

INSIDE BTTRIEVE FILES

68

by Douglas Reilly

Knowing something about Btrieve's undocumented features can make the difference between success and failure when it comes to recovery of damaged Btrieve-compressed records.

EXAMINING ROOM

EXAMINING PC AUDIO

78

by John W. Ratcliff

There's a lot of noise being made about the need for software support for sound. John examines the options available to PC programmers, then presents a sound driver that produces high-quality digitized sound on PCs without requiring any extra hardware.

PROGRAMMER'S WORKBENCH

PROXY: A SCHEME-BASED PROTOTYPING LANGUAGE

86

by Burt Leavenworth

Proxy, a Scheme-based interactive language with a C-like syntax, provides all the high-level data structures—sets, maps, sequences, and objects—useful for software design and prototyping. In addition to showing you how the language can be used in a typical prototyping session, Burt gives you the Proxy interpreter.

COLUMNS

PROGRAMMING PARADIGMS

109

by Michael Swaine

The basic idea behind visual programming is, "What you see is what you'll make." Michael examines this paradigm, using Serious's development tools for the Macintosh.

C PROGRAMMING

113

by Al Stevens

More D-Flat++ control classes are presented this month, in particular those that provide support for pop-down menus, dialog boxes, radio and command buttons, the check box, and the base class for buttons.

STRUCTURED PROGRAMMING

119

by Jeff Duntemann

Jeff looks at client/server database management and examines APIs ranging from SQL to DILs. He then speculates on what kind of database support Pascal vendors should begin thinking about.

UNDOCUMENTED CORNER

129

edited by Andrew Schulman

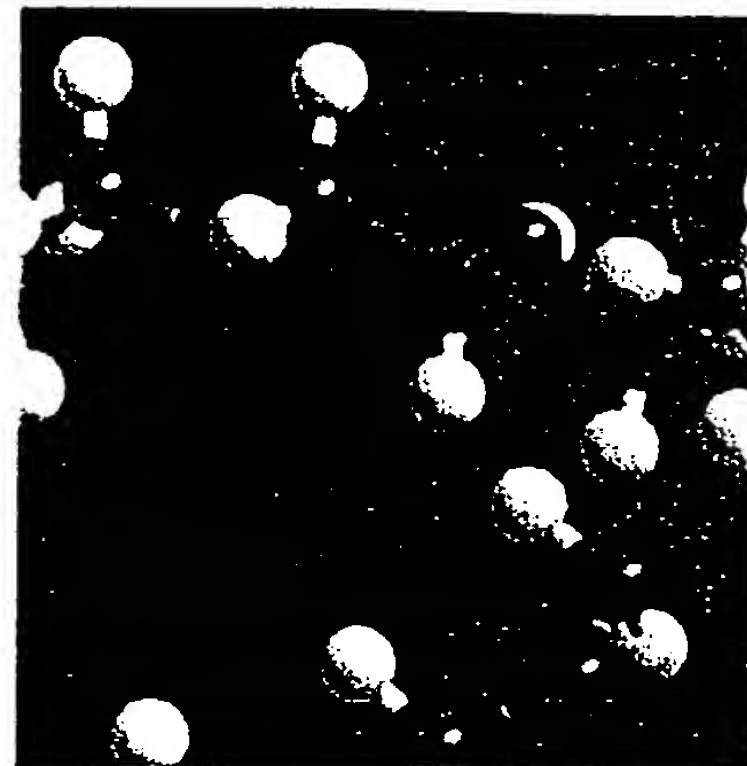
In his inaugural column, Andrew presents Joe Newcomer and Bruce Horn's analysis of the undocumented RGNOBJ structure in Microsoft Windows. The region feature, which is maintained by the Windows GDI, is an arbitrarily-bounded area that can be used for filling, outlining, and clipping.

PROGRAMMER'S BOOKSHELF

139

by Ray Valdes

Ray examines books that focus on artificial life: *Artificial Life II, Emergent Computation*, and John Holland's 1992 edition of *Adaptation in Natural and Artificial Systems*.



FORUM

EDITORIAL

8

by Jonathan Erickson

LETTERS

10

by you

SWAINE'S FLAMES

168

by Michael Swaine

PROGRAMMER'S SERVICES

OF INTEREST

160

by Tami Zemel

SOURCE CODE AVAILABILITY

As a service to our readers, all source code is available on a single disk and online. To order the disk, send \$14.95 (California residents add sales tax) to *Dr. Dobb's Journal*, 411 Borel Ave., San Mateo, CA 94402, or call 1-800-688-3987. Specify issue number and disk format. Code is also available through the DDJ Forum on CompuServe (type GO DDJ), via anonymous FTP from site ftp.mv.com (192.80.84.1) in the /pub/ddj directory, and through M&T Online, a free service accessible via direct dial at 415-358-8857 (8-N-1). (For M&T Online support, call 415-358-9500, extension 220.)

NEXT MONTH

Arriving at the *right* algorithm can be the most creative and enjoyable part of programming—and a properly chosen algorithm can make the difference between a program's success or failure. In April, we'll examine a bevy of challenging, intriguing, and useful algorithms.

Julian and Gregorian Calendars

Date-conversion functions for yesterday and today

Peter J.G. Meyer

The Western calendrical system—known as the Julian calendar and consisting of a year of 12 months and of 365 days with an extra day every fourth year—was established by Julius Caesar (following the advice of the Alexandrian astronomer Sosigenes) in 46 B.C. The extra day may not have been added consistently until A.D. 8, during the reign of Augustus. Subsequently, this calendar became widespread as a result of the expansion of the Roman Empire. The system of numbering years Anno Domini was instituted in A.D. 525 by the Roman abbot Dionysius Exiguus.

The Julian calendar assumes that the average length of a year is 365 days and six hours (since one day is added every four years). The length of the year assumed in the Julian calendar exceeds the current true value by about 11 minutes, resulting in an error of about three days every 400 years. Thus, as the centuries passed the Julian calendar became increasingly inaccurate with respect to the solar year as defined in terms of the solstices and the equinoxes. This was especially troubling to the Church because it affected the determination of the date of Easter, which by the sixteenth century was slipping gradually into summer. To resolve these problems, the calendar

was reformed in 1582 on the authority of Pope Gregory XIII, and the modified calendar is called the Gregorian calendar.

In this article, I'll present a C function which converts any date within an 11-million-year period in either the Gregorian calendar or the Julian calendar into a unique *long int*, a number in the range of approximately -2,000,000,000 through 2,000,000,000. A function is also given for conversion of a *long int* back into a date in one of the calendars. This permits conversion between dates in the Julian and Gregorian calendars and provides a basis for other date-manipulation functions. The date-conversion functions given in this article are used in a general C-function library that I developed, the Dolphin C Toolkit.

Universal Date Conversion

According to the Gregorian reform, ten days (or more exactly, dates) were omitted from the calendar. It was decreed that the day following October 4, 1582 (which was October 5, 1582 in the old calendar) would thenceforth be known as October 15, 1582. In addition, the rule for leap years was changed. In the Julian calendar, a year is a leap year if it is divisible by 4. In the Gregorian calendar, a year is a leap year if it is divisible by 4, with the added criterion that years divisible by 100 must also be divisible by 400. Thus the years 1600 and 2000 are leap years, but 1700, 1800, 1900, and 2100 are not. Finally, it was decreed that new rules for the determination of the date of Easter would be adopted.

Day Numbers

Astronomers use a system of numbering days called Julian-day numbers. The term "Julian-day number" (unlike the term "Julian calendar") does not derive from the name of Julius Caesar. This numbering system is said to have been named after Julius, the father of its inventor. The astronomical system of Julian-day numbers should not be confused with the simpler system of the same name, which associates a date with the number of days elapsed since January first of the same year (according to which December 31, 1993 is Day 365).

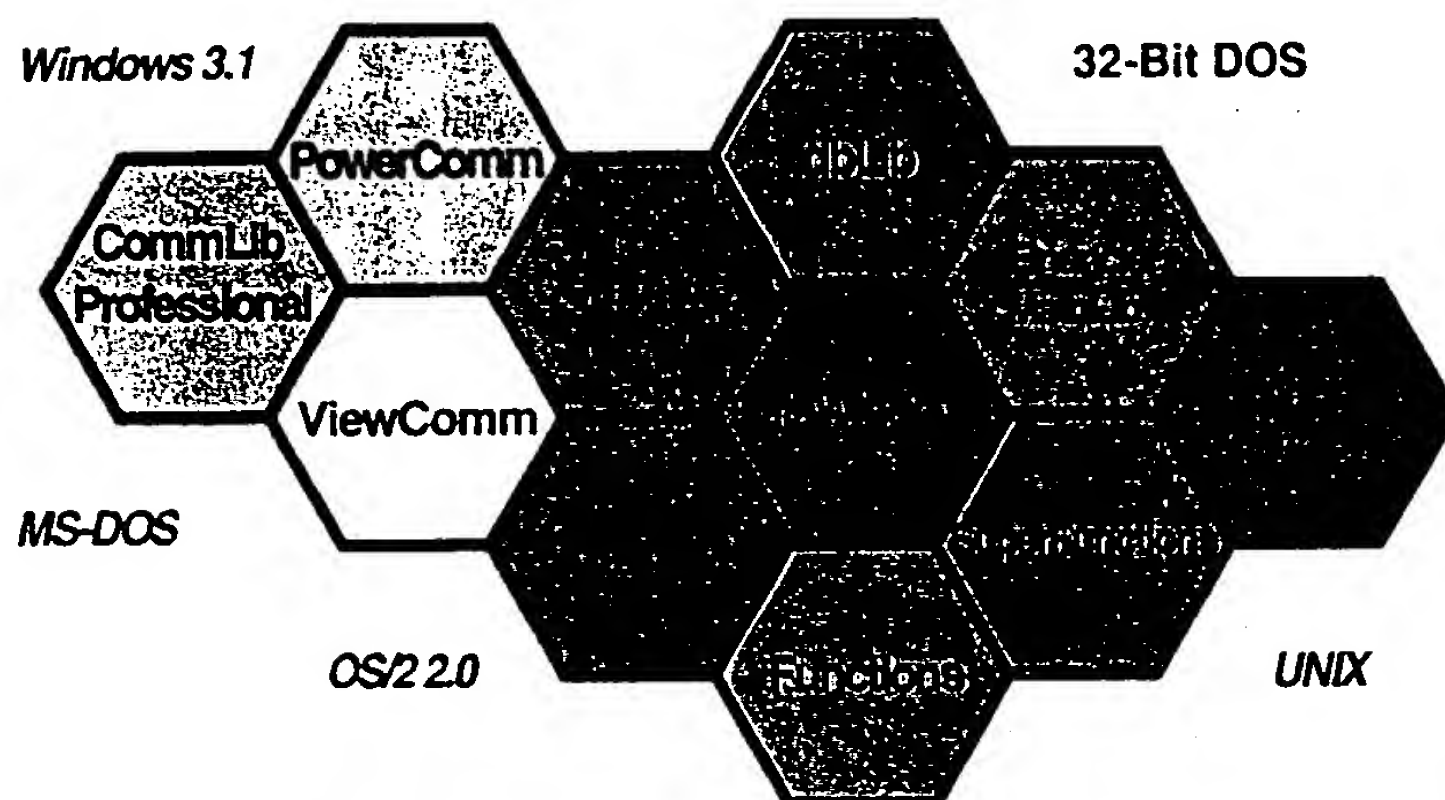
The astronomical Julian-day number of the day specified by a date in either the Julian or the Gregorian calendar is the number of days elapsed from the day designated as January 1, 4713 B.C. in the Julian proleptic calendar. (See the textbox entitled, "The Proleptic Calendars" for more information.) Thus, the Julian-day number of 1/1/-4712 (J) is 0. Note that 4713 B.C. is the year -4712. The Julian-day number of 10/10/1992 is 2,448,906.

There is a simple relationship between Gregorian-day numbers, as used in this method of date conversion, and Julian-day numbers. Given a Gregorian-day number, the corresponding Julian-day number is obtained by adding 2,209,161, the Julian-day number of October 15, 1582).

Listing One, page 158, contains the header file, DATECONV.H, used by the date-conversion functions presented in this article. A structure *Date* contains values for day, month, and year, plus a value *gdn* (for Gregorian-day number, as explained shortly) and a flag indi-

Peter, who currently works in AI research, is the author of the Dolphin C Toolkit and Dolphin Encrypt. He can be reached at 4815 W. Braker Lane, #502-111, Austin, TX 78759 or via the Internet at meyer@mcc.com.

The Greenleaf Collection of C Tools



NEW & HOT!

Greenleaf CommLib 4.0 **NEW VERSION**

CommLib 4.0 contains over 300 robust asynchronous communications functions including XMODEM, YMODEM, ZMODEM, Kermit and ASCII file transfer, XON/XOFF, RTS/CTS, and DTR/DSR handshaking, smart modem control, interrupt driven serial I/O for 13 drivers using consistent "Level 2" API for all drivers. Supports DOS, 286 Extended DOS, and Windows 3.1. \$359.

Greenleaf PowerComm **NEW**

PowerComm Toolkit for Windows is a native 32-bit Windows 3.1 implementation of Greenleaf CommLib v4.0. Extends CommLib capability using DLL and virtual device driver (VxD) to put time sensitive interrupt service routines at "Ring 0" for optimal throughput. Supports multiple Windows and/or DOS applications and normal and most standard multi-port hardware. \$179 (requires CommLib).

Greenleaf CommLib Professional **NEW**

New standard asynchronous communications toolkit for Windows, DOS, and 286 Extended DOS. Includes Greenleaf CommLib 4.0 and PowerComm Toolkit for Windows. \$538.

Greenleaf Comm++ 2.0 **NEW VERSION**

Comm++ class library includes numerous classes for asynch communications—from elemental hardware control classes to file transfer and terminal emulations include VT100, VT52, and ANSI. XMODEM, YMODEM, ZMODEM, Kermit, and ASCII file transfers. Many handshaking options. Screen drivers. Supports numerous standard and intelligent multiport boards for DOS, Windows 3.x, and OS/2 2.0. \$249.

Greenleaf ViewComm

Lets you see data and status on the line + timestamps. Capture to buffer or file. Review using ASCII, hex, decimal, octal, or EBCDIC display. Extensive triggers: modem signal and string match criteria to start or stop capture. Source Mode lets you send characters from keyboard, file, or both. Configurable, push-to-DOS, many more features. \$399.

Greenleaf Functions

Over 370 functions including DOS interface, keyboard, mouse, color text, graphics, screen & video, string, time and date, polled-mode serial, interrupt services and Ctrl-Break, equipment interface, and extensive printer support. \$229.

Greenleaf SuperFunctions

SuperFunctions includes (over 370 functions) windows and basic menus (from DataWindows), Expanded Memory support, extensive mouse and keyboard support, DOS Critical Error Handler support, advanced DOS functions, an exhaustive set of time and date manipulation functions including project scheduling support, etc. No overlap with Greenleaf Functions. \$299.

dBLib / SoftC Database Library 3.2

SoftC Database Library now published by Greenleaf. Reviewers rave about the speed and flexibility of this extensive "xBASE" interface library. Supports dBASE III+ & IV, FoxBASE+, FoxPlus, dBase, Quicksilver, and Clipper, including their proprietary index and memo file systems. Rated fastest and most compatible with C language. Complete ISAM database toolkit. Highly portable ANSI C dBLib source supports DOS, Windows 3.1 (DLL included), UNIX, and OS/2. \$249.

Greenleaf DataWindows 3.0

DataWindows 3.0 with MakeForm data entry form generator, a true WYSIWYG designer to create resource files. Changes to forms do not require recompilation! 400 functions for logical windows, transaction oriented data entry, and extensive menus. 13 installable state-change functions including field and form validation. \$350.

Greenleaf Financial MathLib

A unique 19 digit "DECimal" number system avoids floating point errors. Over 195 functions include arithmetic, error processing, test, array math, transcendentals, percentages, I/O, simple & compound interest, loan amortization, discounted cash flow, annuities, actuarial and general business functions, bond calculations, depreciation schedules, date manipulations, statistics, linear estimation. Modeled on the HP-12C calculator. \$395.

Greenleaf Financial MathLib++

MathLib++ is a C++ version of Financial MathLib described above. C++ iostreams are fully supported with overloaded I/O and math operators, so that programming is greatly simplified. One function call gets a complete loan amortization table. Another computes the Internal Rate of Return of a set of discounted cash flows. \$199.

All Greenleaf Products come with reference manual, online documentation system (with free engine), FREE unlimited phone support, FREE BBS access, newsletter, extensive compilable examples for each function, NO Royalties, 60-day money-back guarantee, and are backed by the industry's most experienced tech support staff. Greenleaf Gold Support (extra annual fee) puts upgrades on your desk as soon as they are released and gives U.S. customers toll-free access to tech support and BBS.

Call Now:

800-523-9830

214-248-2561

FAX: 214-248-7830

Greenleaf Software, Inc.

16479 Dallas Parkway, Suite 570



GREENLEAF

(continued from page 152)

ating the validity of the date. A day/month/year value is ambiguous until the particular calendar is specified. A date is completely specified using an instance of the structure together with an instance of a separate *char* variable that has the value G or J.

We first need to ascertain whether a given year is a leap year (in the Julian or Gregorian calendars). Functions to do this are given in DATECONV.C (see Listing Two, page 158). A universal date-integer conversion method, as understood here, consists of two functions: The first takes a date (either Julian or Gregorian) and returns a positive or negative integer (*long int*); the second takes a *long int* and a calendrical specification (G or J) and returns a date in that calendar. It does not matter which date corresponds to day 0 as long as there is a quickly computable, one-to-one correspondence between dates in a calendar and numbers.

The method presented here converts dates into the number of days before or after October 15, 1582—the day that the Gregorian calendar came into effect. Thus, October 15, 1582 (Gregorian) corresponds to day 0; October 16, 1582 (Gregorian) to day 1; and Oc-

tober 14, 1582 (Gregorian) to day -1. The number corresponding to a date is thus called the "Gregorian-day number." Dates in the Julian calendar, as well as those in the Gregorian calen-

*The length of the
year assumed in the
Julian calendar
exceeds the current
true value by about
11 minutes*

dar, are mapped into Gregorian-day numbers. Thus, the day preceding October 15, 1582 in the Gregorian calendar is both October 4 in the Julian calendar and October 14 in the Gregorian calendar—both have Gregorian-day number -1.

The code for the function to convert a date in one of the calendars to a Gregorian-day number, and for the func-

tion to convert a Gregorian-day number to a date in a specified calendar, is given in Listing Two. The *Date* structure uses *long int* variables (signed) for the year and the Gregorian-day number. The largest integer representable in a signed *long int* is 2,147,483,647. Due to the method of calculation, however, the largest Gregorian-day number that can be used is 2,146,905,911. This corresponds to the dates July 11, 5,879,611 (Gregorian) and October 19, 5,879,490 (Julian). By that time, dates in the Gregorian and Julian calendars will differ by about 121 years. (This will be of no practical importance since by that time both calendars will likely have been superseded.)

To use the conversion functions, declare a structure of type *Date* (defined in Listing One) and pass to the functions a pointer to the structure along with a calendrical specification (G or J). If you are converting from Gregorian-day number to date, define the *gdn* structure variable before calling *gdn_to_date()*. Conversely, define the variables *day*, *month*, and *year* before calling *date_to_gdn()*. On return from the functions, extract either *gdn* or the date values from the structure. Before using the *gdn* value, it is advisable to check the

Adopting the Gregorian Calendar

There was no necessity for 10 days; rather than, say, 12 days to have been omitted from the calendar. In fact, the calendar could have been reformed so as to keep the year in step with the seasons without omitting any days at all, since only the new rule for leap years is required to keep the calendar synchronized with the equinoxes. In fact, ten days were omitted in order to fix the date for the Spring equinox at March 21, which was the date of the equinox at the time of the Council of Nicea in the fourth century.

Upon the promulgation of Pope Gregory's decree, the Gregorian calendar was adopted immediately in Italy, Spain, Portugal, and Poland, and shortly thereafter in France and Luxembourg. During the next two years, most Catholic regions of Germany, Belgium, Switzerland, and the Netherlands came on board. Hungary followed in 1587. The rest of the Netherlands, Denmark, Germany, and Switzerland made the change in 1699-1701.

By the time the British were ready to acquiesce, the old calendar had drifted off by one more day, requir-

ing a correction of 11 days, rather than 10, to locate the Spring equinox (usually) at March 21. The Gregorian calendar was adopted in Britain (and in the British colonies) in 1752, with September 2, 1752 being followed immediately by September 14, 1752.

In many countries, the Julian calendar was used by the general population long after the official introduction of the Gregorian calendar. Thus, events were recorded in the sixteenth to eighteenth centuries with various dates, depending on which calendar was used. Dates recorded in the Julian calendar were marked "O.S." for "Old Style," and those in the Gregorian calendar were marked "N.S." for "New Style."

To complicate matters further, the first day of the year was celebrated in different countries and regions on January 1, March 1, March 25, or December 25. With the introduction of the Gregorian calendar in Britain and the American colonies, people ceased to celebrate New Year's Day on March 25, as had been their custom, and instead began to celebrate it on January 1. Previously, March 24 of one year had been followed by March 25 of the

next year. Thus George Washington's birthday, which was 2/11/1731 O.S., became 2/22/1732 N.S.

Sweden adopted the Gregorian calendar in 1753, Japan in 1873, Egypt in 1875, China and Albania in 1912, Bulgaria in 1915 or 1916, Romania in 1919, and Turkey in 1927. Following the Bolshevik Revolution in Russia, it was decreed that the day following January 31, 1918 O.S., would become February 14, 1918 N.S.

In 1923, the Eastern-Orthodox church adopted a modified form of the Gregorian calendar. Whereas in the Gregorian calendar a century year is a leap year only if division by 4 leaves a remainder of 1, 2, or 3, in the Eastern system a century year is a leap year only if division by 9 leaves a remainder of 2 or 6. This renders the calendar slightly more accurate. October 1, 1923 in the Julian calendar became October 14, 1923 in the Eastern-Orthodox calendar. The date of Easter is determined by reference to modern lunar astronomy (in contrast to the more approximate, rule-based, lunar model of the Gregorian system).

—P.M.

UNIX & Open Systems

COMMUNICATIONS

HARDWARE

SOFTWARE

INTEROPERABILITY

ENTERPRISE COMPUTING

DESKTOP

INTEGRATED SYSTEMS

PERIPHERALS

CLIENT/SERVER APPLICATIONS

NETWORKING

SOLUTIONS

It All Works at
UNIForum 1993

MARCH 17-19
MOSCONE CENTER
SAN FRANCISCO

UniForum

UNIForum 1993 is the largest UNIX & Open Systems Event in the world! It's Hot - It's New - It's Answers - It's Now:

- 350 Exhibitors with answers
- 119 New Conference and Tutorial solutions
- On-Line, Real-Time Show Network
- FREE UNIX & Open Systems Introductory Primers
- FREE Exhibitor Presentations
- FREE Keynote and Plenary sessions
- FREE Birds-of-a-Feather end-user gatherings
- Plus 10th Anniversary Conference Surprises!

CELEBRATING OUR

10
TH

ANNIVERSARY CONFERENCE

UNIForum 1993

MARCH 17-19, MOSCONE CENTER, SAN FRANCISCO

FAX THIS FORM TODAY! REGISTER FREE FOR UNIForum 1993 EXHIBITS.

FAX: 1-708-260-0396

Registration Deadline: February 26, 1993

For more information, phone 1-800-323-5155. Outside the U.S. and Canada, 1-708-260-9700.

FIRST NAME										LAST NAME											
TITLE										AREA CODE					TELEPHONE						
COMPANY																					
ADDRESS																					
CITY										STATE		POSTAL/ZIP CODE									
COUNTRY										AREA CODE		FAX									

NOTE: No one under the age of 18 admitted.

1. MY JOB FUNCTION/TITLE (select one only):
- ☐ A President/Principal/Vice President/Director
 - ☐ B Corporate Administration/Financial Officer
 - ☐ C Data Communications/Telecommunications Management
 - ☐ D MIS/DP Management
 - ☐ E Engineering/Technical/Manufacturing/Operations Management
 - ☐ F Sales & Marketing Management
 - ☐ G R&D/Software Development
 - ☐ H Applications/Systems Programmer/Analyst
 - ☐ J Purchasing

- ☐ V Software Developer
- ☐ W Manufacturing (non-computer)
- ☐ X Systems Integrator/Service Provider
- ☐ Y Manufacturer of Computers/Systems/Software/Peripherals
- ☐ Z Other Business

5. MY DEGREE OF PURCHASE INFLUENCE ON INFORMATION PRODUCTS/SERVICES:
- ☐ HH I make the final decision
 - ☐ JJ I recommend and strongly influence the final decision
 - ☐ KK I specify products/services that we need
 - ☐ LL I play no role in the purchasing process

2. PRIMARY BUSINESS OF MY ORGANIZATION (select one only):
- ☐ K Financial Services
 - ☐ L Utilities/Transportation
 - ☐ M Government/Military/Defense
 - ☐ N Engineering/Science/R&D
 - ☐ P Education
 - ☐ Q Medical & Health Services
 - ☐ R Consulting Services
 - ☐ S Wholesale/Retail Trade
 - ☐ T Communications/Telecommunications
 - ☐ U VAR/Dealer/Distributor/OEM/Retailer

3. NUMBER OF EMPLOYEES IN MY ORGANIZATION (all locations):
- ☐ AA 1-50
 - ☐ BB 51-500
 - ☐ CC 501-5000
 - ☐ DD Over 5000

4. NUMBER OF YEARS I HAVE USED UNIX:
- ☐ EE None
 - ☐ FF 1 to 5 years
 - ☐ GG Over 5 years

6. I SPECIFY OR INFLUENCE THE PURCHASE OF (check all that apply):

- ☐ MM Mainframes/Superminis
- ☐ NN Minicomputers/Small Business Computers/Micros/Desktops
- ☐ PP Software (Operating System)
- ☐ QQ Software (Applications)
- ☐ RR Network/Communications Systems
- ☐ SS Office Automation Systems
- ☐ TT LAN Equipment/Systems/Services

7. MY COMPANY'S TOTAL GROSS SALES OR REVENUES:

- ☐ UU Under \$1 million
- ☐ VV \$1-\$24 million
- ☐ WW \$25-\$99 million
- ☐ XX \$100-\$249 million
- ☐ YY \$250-\$499 million
- ☐ ZZ \$500 million and over

☐ Also send me detailed info on Conferences and Tutorials.

Photocopy this form for additional registrants.

(continued from page 154)

validity flag, which will be TRUE if the date passed was a valid date in the specified calendar, and FALSE otherwise. For example, the attempt to convert February 29, 1900 (Gregorian) to a Gregorian-day number will produce a FALSE in the validity variable.

The *lfloor()* function in Listing Two is analogous to the Microsoft floating-point library *floor()* function, and overcomes a small problem in integer arithmetic. The date-conversion functions described earlier need a long-integer division operation such that, for all long integers *a* and *b*, *a/b* is the largest integer not greater than the real number *a/b*. MSC's division operator produces this result if *a* and *b* are positive, but not if *a* is negative and *b* is positive. The *lfloor()* function provides what is needed.

Finally, DATECONV.C also contains a function to convert a Gregorian-day number into a day of the week. This is independent of the particular calendar used.

Testing

No function or program can be relied upon unless it is tested thoroughly. The program DATETEST.C in Listing Three (page 159) tests the functions in Listing

*The astronomical
Julian-day number
is the number of
days elapsed from
January 1, 4713 B.C.*

Two. DATETEST takes two numbers, *n* and *m*, on the command line and performs conversions for *n*, *n+m*, *n-m*, *n+2*m*, *n-2*m*, and so on, up to the largest integer that can be handled. It first converts the number to a date using *gdn_to_date()*, then converts the

resulting date back to a number using *date_to_gdn()*. The program does this for both the Gregorian and the Julian calendars. If the number resulting from converting a number to a date and back to a number were different from the initial number, then a bug would be revealed.

If DATETEST is run with command-line parameters 0 and 1, then all dates forward and backward from October 15, 1582 (Gregorian) are tested, although the program displays the results only for every *N*th conversion. (*N* is currently defined as 3000 in Listing Three.)

Since there are over four million input numbers that could be tested, exhaustive testing is not practical unless you can run the program on a very fast computer. It has been shown, however, that when using DATETEST 0 1, no bugs are revealed for all Gregorian-day numbers in the range -14,235,000 through 14,235,000. The corresponding dates include all dates in both calendars for all years from -37,390 to 40,555.

Derivative Calendrical Functions

Given the basic date-number conversion functions, it is not difficult to develop other calendrical functions. In fact, there are 38 date functions in the Dolphin C Toolkit, including functions to determine which of two dates is earlier, how many days separate two dates, and how many weekdays separate two dates. There is also a function to format a date in hundreds of different ways.

Conclusion

The Dolphin C Toolkit includes a demonstration program, CAL_FNS, which allows conversion between dates and Julian-day numbers. This program also provides the day of the week of any date and the number of days between two dates. CAL_FNS allows us to determine, for example, that the storming of the Bastille occurred on a Tuesday. We can also discover that on the evening of April 18, 1521, when Luther defended himself against charges of heresy before the Holy Roman Emperor, Charles V, it was a Thursday. Finally, the coronation of Charlemagne as Holy Roman Emperor in Rome on Christmas day, 800, occurred on a Friday. We may reasonably suppose that festivities continued throughout the weekend.

DDJ

(Listings begin on page 158.)

Vote for your favorite feature/article.
Circle Reader Service No. 15.

The Proleptic Calendars

Every date recorded in history prior to October 15, 1582 (Gregorian), such as the coronation of Charlemagne as Holy Roman Emperor on Christmas day in the year 800, is a date in the Julian calendar, since prior to those dates the Gregorian calendar had not yet been invented. We can, however, identify particular days prior to October 15, 1582 (Gregorian) by means of dates in the Gregorian calendar simply by projecting the Gregorian dating system back beyond the time of its implementation. A calendar obtained by extension earlier in time than its invention is called "proleptic."

For example, although the Gregorian calendar was implemented on October 15, 1582 (Gregorian), we can still say that the date one year before was October 15, 1581 (Gregorian), even though people alive on that day would have said that the date was October 5, 1581 (the Julian date at that time). As another example, the date of the coronation of Charlemagne was December 29, 800 in the Gregorian proleptic calendar.

Similarly, dates after October 15, 1582 (Gregorian) have equivalent, but different, dates in the Julian calendar. For example, this article was completed on October 10, 1992 in the Gregorian calendar, but we could

equally well say that it was completed on September 28, 1992, in the Julian calendar. As another example, the date of the winter solstice in the year 2012 is December 21, 2012 (Gregorian), which is December 8, 2012 (Julian).

Thus, any day in the history of the Earth, either in the past or in the future, can be specified as a date in either of these two calendrical systems. The dates will generally be different. In fact, they will be the same only for dates from March 1, 200 to February 28, 300. The dates in neither calendar will coincide with the seasons in the distant past or distant future, but that does not affect the validity of these calendars as systems for uniquely identifying particular days.

Astronomers designate years B.C. by means of negative numbers. In order to avoid a hiatus between the year 1 and the year -1, there has to be a year 0. Thus, astronomers adopt the convention: A.D. 1 is equal to Year 1; 1 B.C. is equal to Year 0; 2 B.C. is equal to Year -1; and so on. More generally, a year popularly designated *n* B.C. is designated by astronomers as the year $-(n-1)$. Finally, the rules for leap years in both calendars are valid for the year 0 and for negative years, as well as for positive years.

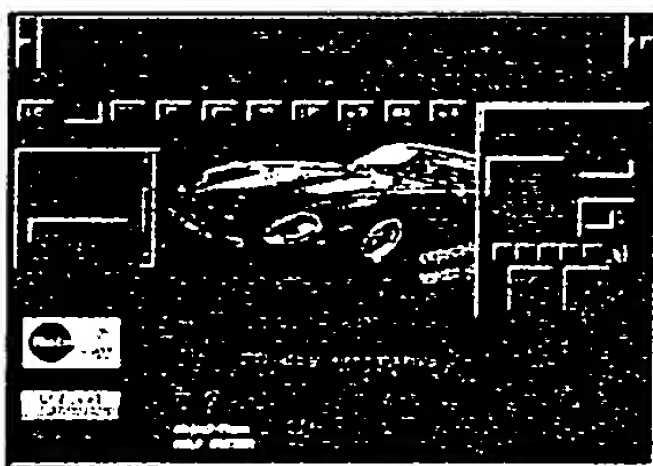
—P.M.

PRODUCT SHOWCASE

Programming Power for DOS and Windows

RSN 1251

The object-menu application framework unleashes the full power of object-oriented C++ for your DOS graphics or Microsoft Windows application. Effortlessly creates windows, menus, dialogs, data entry, combo box, spin control and more. Flexible styling options including Motif. Over 100 example programs. Free demo. \$129 - \$898.



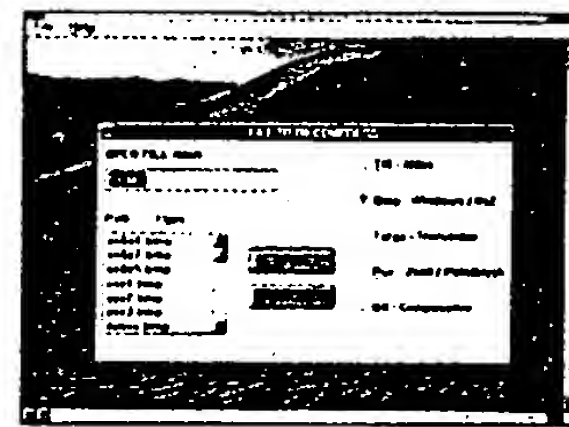
Island Systems

7 Mountain Road
Burlington, MA 01803
Phone: (617) 273-0421 FAX: (617) 270-4437

TIFF, PCX, TARGA, GIF, DIB, BMP, DCX, EPS, WMF, WPG, PICT, JPEG AccuSoft Image Format Library 3.0 DOS and Windows No Royalties

RSN 1254

Import, export, convert, display, and print all above formats! Includes several sample programs with source code. Supports all languages. **Format compatibility guaranteed!** Rotate, scale, color reduction, sharpen etc. 30 day satisfaction guarantee. \$495



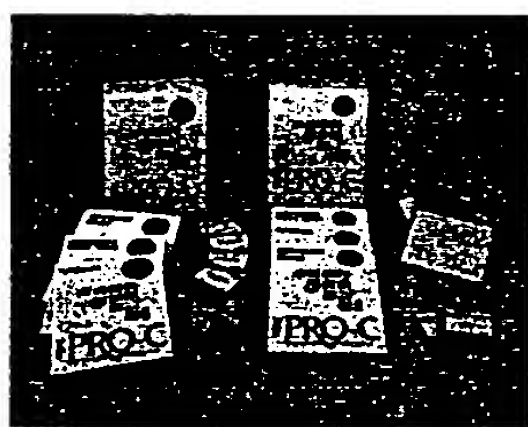
AccuSoft Corporation

P.O. Box 1261
Westboro, MA 01581
(800) 525-3577 (508) 898-2770 FAX (508) 898-9662

PRO-C

RSN 1252

Family of menu-driven C code generators that automate production of information systems containing menus, screens, reports, and batch processing. Supports most popular file handlers, and bundled with PRO-Tree, PROdbx, and Btrieve. Library source included. Royalty free and no runtime required.



Pro-C Limited

Suite 551, 180 King St. South
Waterloo, ON, Canada N2J 2P8
Tel (519) 725-5143 Fax (519) 571-1504

Hipparchus™ GIS Enabling Technology

RSN 1255

Over 200 functions provide C developers with breakthrough GIS capabilities. Interface with any GUI and any DBMS. Ellipsoidal vector algebra enables seamless global coverage, lightning-fast spatial indexing, polygon overlay and more!

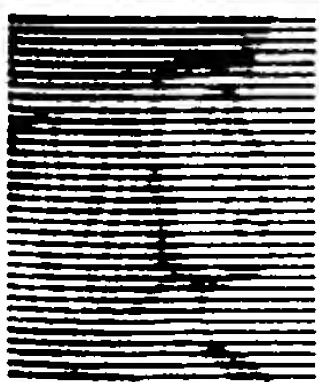


DOS/OS 2 Library, Application Prototyper, Tutorial (270 pp), Reference (340 pp): \$475.

Geodysey Limited

300, 815 Eighth Avenue SW,
Calgary, Alberta, Canada T2P 3P2
Tel 403-234-9848 Fax 403-266-7117

Booth #702



ALDUS

Join us at the
Aldus Developers Conference!
April 21-22-23, 1993
Seattle, Washington

RSN 1253

Explore exciting development opportunities with Aldus and the professional graphics industry at our International Developers Conference. Participate in hands-on workshops, seminars, and product demos. Learn about Aldus Additions. Meet with Aldus engineers and marketing representatives.

For more information contact:
Developers Desk • Aldus Corporation
411 First Avenue South, Dept. DD
Seattle, WA 98104
(206) 343-4249 • Fax (206) 343-4240

Source Code on CD-ROM

RSN 1256

**386BSD & LINUX
X11R5
All GNU Sources
2nd Berkeley Networking Tape
and much more!**



P.O. Box 338, Pennington, NJ 08534
(800) 800-6613 • (609) 683-5501 • Fax: (609) 683-5502
info@infomagic.com

CALENDARS

Listing One (Text begins on page 152.)

```
/* DATECONV.H -- Header file for date functions -- Last mod.: 1992-10-10 */
#define TRUE 1
#define FALSE 0
typedef struct
{
    int day;
    int month;
    long year;
    long gdn; /* gdn and valid are for internal use */
    int valid; /* by functions in this library */
} Date;

/* declarations of functions defined in DATECONV.C */
void date_to_gdn(Date *dt, char calendar);
void gdn_to_date(Date *dt, char calendar);
long lfloor(long a, long b);
int is_leap_year_c(long year, char calendar);
int is_leap_year(long year);
void set_feb_length(long yr, char calendar);
void reset_feb_length(void);
int day_of_week(long gdn);
```

Listing Two

```
/* DATECONV.C -- Universal date conversion functions -- Last mod.: 1992-10-10 */
#include <STDLIB.H> /* for labs() */
#include <CTYPE.H>
#undef toupper /* so as to use function, not macro */

#include "DATECONV.H"
int month_length[14] =
{
    0, 31, 28, 31, 30, 31, 30,
    31, 31, 30, 31, 30, 31, 0
};

/* months 0 and 13 have 0 days */
/* the calendar parameter is always 'G' or 'J': G = Gregorian, J = Julian */
int is_leap_year_c(long year, char calendar)
{
    calendar = (char)toupper((int)calendar);
    if (year%4) /* if year not divisible by 4 */
        return (FALSE);
    else
    {
        if (calendar == 'J')
            return (TRUE);
        else /* calendar == 'G' */
            return ((year%100 != 0L || year%400 == 0L) ?
                TRUE : FALSE);
    }
}
```

```
int is_leap_year(long year)
{
    return (is_leap_year_c(year, 'G'));
}

void set_feb_length(long yr, char calendar)
{
    month_length[2] = 28 + is_leap_year_c(yr, calendar);
}

void reset_feb_length(void)
{
    month_length[2] = 28;
}

/* function to convert date to Gregorian day number sets valid flag to FALSE
 * if date invalid, otherwise returns number of days before or after the day
 * the Gregorian calendar came into effect (15-OCT-1582) */
void date_to_gdn(Date *dt, /* dt is a pointer to a structure */
    char calendar) /* 'G' or 'J' */
{
    int day = dt->day;
    int month = dt->month;
    long year = dt->year;
    long gdn;

    calendar = (char)toupper((int)calendar);
    set_feb_length(year, calendar);
    if (month < 1 || month > 12
        || day < 1 || day > month_length[month])
        dt->valid = FALSE;
    else
    {
        /* calculate number of days before/after October 15, 1582 (Gregorian) */
        gdn = (year-1)*365 + lfloor(year-1.4L);
        if (calendar == 'G')
            gdn += lfloor(year-1.400L) - lfloor(year-1.100L);
        while (--month)
            gdn += month_length[month];
        gdn += day - 57736L - 2*(calendar=='J');
        dt->gdn = gdn;
        dt->valid = TRUE;
    }
    reset_feb_length();
}

/* function to convert gregorian day number to date */
void gdn_to_date(Date *dt, char calendar)
{
    int month, i, exception;
    long year, gdn, y4, y100, y400;
    calendar = (char)toupper((int)calendar);
    gdn = dt->gdn;
    gdn += 57735L + 2*(calendar=='J');
    y400 = 146100L - 3*(calendar=='G');
    y100 = 36525L - (calendar=='G');
```

DR. DOBB'S JOURNAL

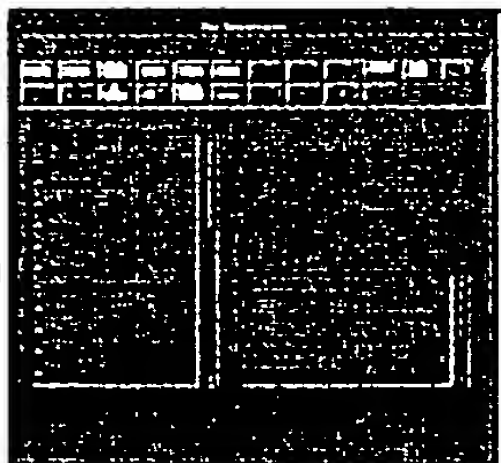
PRODUCT SHOWCASE

Zinc Application Framework 3.5

Create object-oriented, multiplatform applications with Zinc Application Framework 3.5. Zinc's flexible, C++ architecture supports Microsoft Windows and Windows NT, OS/2 2.0 and UNIX Motif as well as DOS Graphics and DOS Text—all with one set of source code. Zinc Designer runs in native mode on each target platform. Zinc POST™ (Persistent Object Storage Technology) provides robust platform-independent resources. Full source included.

Zinc Software Incorporated

405 South 100 East, 2nd Floor
Pleasant Grove, Utah 84062
800-638-8665 VOICE 801-785-8996 FAX 801-785-8997 BBS



RSN 1257

RISC Design-Made-Easy Free Fusion29K™ Catalog!

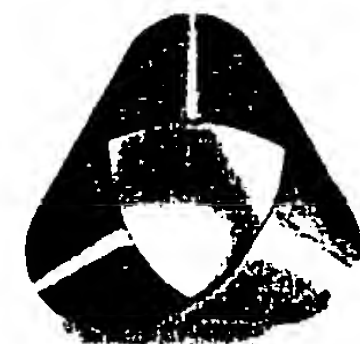
AMD provides extensive application development support for the 29K™ Family with the Fusion29K Partner Program, the industry's first and most comprehensive support system for embedded RISC designers. The Program offers designers all the industry-standard, application-specific software and hardware tools they need to speed their products to market.

The Fusion29K Catalog includes:

- Silicon and prototyping support
- Software generation and debug tools
- System development
- Real-time systems
- Board level products
- Laser printer, network, and graphics solutions
- Training and custom support

Call for your free Fusion29K Catalog Today!

Advanced Micro Devices, Inc.
5204 E. Ben White Blvd., MS 561, Austin, Texas 78741
(800) 292-9263 ext. 3 or Direct (512) 462-5651 Fax (512) 462-5051



RISC Design-Made-Easy™ Solutions

RSN 1258

```

y4 = 1461L;
exception = FALSE;
year = 400*lfloor(gdn.y400); /* 400-year periods */
gdn -= y400*lfloor(gdn.y400);
if ( gdn > 0L )
{
    year += 100*lfloor(gdn.y100); /* 100-year periods */
    gdn -= y100*lfloor(gdn.y100);
    exception = ( gdn == 0L && calendar == 'G' );
    if ( gdn > 0L )
    {
        year += 4*lfloor(gdn.y4); /* 4-year periods */
        gdn -= y4*lfloor(gdn.y4);
        if ( gdn > 0L )
        {
            i = 0;
            while ( gdn > 365 && ++i < 4 )
            {
                year++;
                gdn -= 365L;
            }
        }
    }
}
if ( exception )
    gdn = 366L;
/* occurs once every hundred years with Gregorian calendar */
else
{
    year++;
    gdn++;
}
set_feb_length(year,calendar);
month = 1;
while ( month < 13 && gdn > month_length[month] )
    gdn -= month_length[month++];
if ( month == 13 )
{
    month = 1;
    year++;
}
reset_feb_length();
dt->day = (int)gdn;
dt->month = month;
dt->year = year;
dt->valid = TRUE;
}
long lfloor(long a,long b) /* assumes b positive */
{
    return ( a >= 0L ? a/b : ( a%b == 0L ) ? -1 - labs(a)/b );
}
/* labs() returns the absolute value of its long int argument */
}
/* returns day of week for given Gregorian day number; 0=Sunday, 6=Saturday */
int day_of_week(long gdn)
{
    return ((int)((((gdn%7)+12)%7));
}

```

Listing Three

```

/* DATETEST.C -- Tests date conversion routines -- Last mod.: 1992-10-10 */
/* Link with DATECONV.OBJ by using CL DATETEST.C DATECONV.C */

```

```

#include <STDIO.H>
#include <STDLIB.H> /* for exit() */
#include <CONIO.H>

#include "DATECONV.H"

void main(int argc, char **argv):
void display(long n):
void test(long n, char calendar):

Date date;

#define ESCAPE '\x1B'
#define N 3000
#define MAXIMUM_GDN 2146905911
/* largest Gregorian day number that can be handled */
void main(int argc, char **argv)
{
    long n, m;
    if ( argc < 3 )
    {
        printf("Syntax: DATETEST start increment\n");
        return;
    }
    n = atol(argv[1]); /* number to start with */
    m = atol(argv[2]); /* increment */
    printf("Quit with Escape.");
    while ( TRUE )
    {
        if ( n%(N*m) == 0 )
        {
            display(n);
            display(-n);
        }
        test(n,'G');
        test(n,'J');
        test(-n,'G');
        test(-n,'J');
        if ( n > MAXIMUM_GDN-m )
        {
            printf("\ngdn = %ld",n);
            return; /* since next n is too large */
        }
        n += m;
        if ( kbhit() )
        {
            if ( getch() == ESCAPE )

```

```

        break;
    }
}
void display(long n)
{
    date.gdn = n;
    printf("\ngdn = %ld\n",n);
    gdn_to_date(&date,'G');
    printf(" %02d/%02d/%ld (G)",date.month,date.day,date.year);
    gdn_to_date(&date,'J');
    printf(" %02d/%02d/%ld (J)",date.month,date.day,date.year);
}
void test(long n,char calendar)
{
    date.gdn = n;
    gdn_to_date(&date,calendar);
    if ( !date.valid )
    {
        printf("\ngdn = %ld %d/%d/%ld (%c) Date invalid!",
            date.gdn,date.month,date.day,date.year,calendar);
        exit(1);
    }
    date.gdn = 0L;
    date_to_gdn(&date,calendar);
    if ( date.gdn != n )
    {
        printf("\ngdn = %ld %d/%d/%ld (%c) n = %ld!",
            date.gdn,date.month,date.day,date.year,calendar,n);
        exit(1);
    }
}

```

End Listings

GRAF/DRIVE PLUS 3.0

Turbo Graphics for Printers, Plotters—and now DTP Files

Publication-quality hard copy graphics for Turbo C & Pascal! Install our BGI printer drivers and your screen graphics program prints graphs too. Add our new DTP file formats and import the graphs to your desktop publisher or word processor. Great for user manuals!

- LaserJet, Epson, DeskJet, DeskJet 500C, PostScript, ProPrinterX24, PaintJet, HPGL plotters, and more.
- PCX, TIF, CGM, WPG, DXF, Video Show, color dot matrix printers, and more (Supplemental Driver Disk).
- Arc, bar, bar3d, circle, fillpoly, line, ellipse, pieslice, line/fill patterns, etc—all 100% interchangeable with Turbo's screen graphics. Plot to LPT, COM, or disk.
- Use PostScript fonts and LJ soft fonts (including LJIII scalable fonts), plus Borland stroke fonts.
- Not a screen dump. You get the full printer resolution.
- Not TSR. Loads on the heap at runtime. EMS optional.
- Portrait and landscape. Control image size/location.
- Compatible with Graphics Menu, Zinc, Turbo Vision.
- "Highly recommended." - Jeff Duntzman, Dr. Dobbs, 11/90. Our customers include Harvard, UCLA, HP, TRW, and Los Alamos Labs.

For Turbo C 2.0, C++, Pascal 5-6. Personal License \$149, Developers with royalty-free distribution \$299. Supplemental DTP drivers \$99. 30-day m/b guarantee.

FLEMING SOFTWARE BOX 569 OAKTON, VA 22124
(703) 591-6451

CIRCLE NO. 456 ON READER SERVICE CARD

SPECIAL ISSUE

PC MAGAZINE

SEPTEMBER 25, 1990

THE INDEPENDENT GUIDE TO IBM-STANDARD #

VOLUME 9 NUMBER 16

FIRST LOOKS

NetWare and LAN Manager Upgrades: A New Pecking Order?

FIRST LOOKS

SuperPrint Solves Slow Windows Printing

MAINTENANCE

What You Need to Know Before You Sign a Service Contract

AFTER HOURS

Better than the Book: 3 Encyclopedias on CD-ROM Deliver Multimedia Today

First Ever!
SERVICE AND RELIABILITY SURVEY

We Rate the Industry on Reliability, Tech Support, Repair Service, and Customer Loyalty

- ☒ 27 Computer Vendors
- ☒ 22 Monitor Makers
- ☒ 17 Printer Companies
- ☒ 12 Hard Disk Manufacturers
- ☒ 12 Laptop Vendors
- ☒ 11 Graphics Board Makers

#534H*****5-DIGIT 47906
00PR00000094 9#
405384 SPR 00000094 1610
PURDUE UNIV NOV 06 91
PCF KARDEX USAB

PC SURVEY

Dear Reader:

We know how much information you can provide to improve our products and services. Please write in the space provided.

Please add any comments about the computer equipment you mentioned in this survey. Please write in the space provided.

My experiences with this company ranged from extreme satisfaction to frustration. I wish I had been more satisfied with the manufacturer before I bought this product.

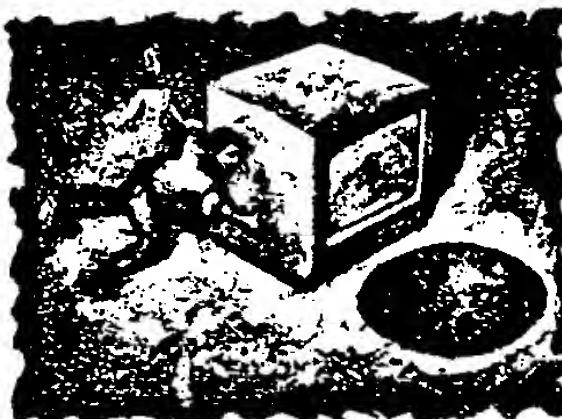
Classification of the following questions and for classification of the following questions:

- Are you personally responsible for the purchase of the equipment? (Yes, on formal basis; No, on informal basis)
- Have you personally opened up the equipment? (Yes, No)
- Which of the following internal components did you personally open up? (Memory, Hard Drive, Power Supply, etc.)
- What is the level of education you have? (High school, college, graduate work but did not (yet) receive a degree (MA, PhD, MD), etc.)

For your help! Now, please...

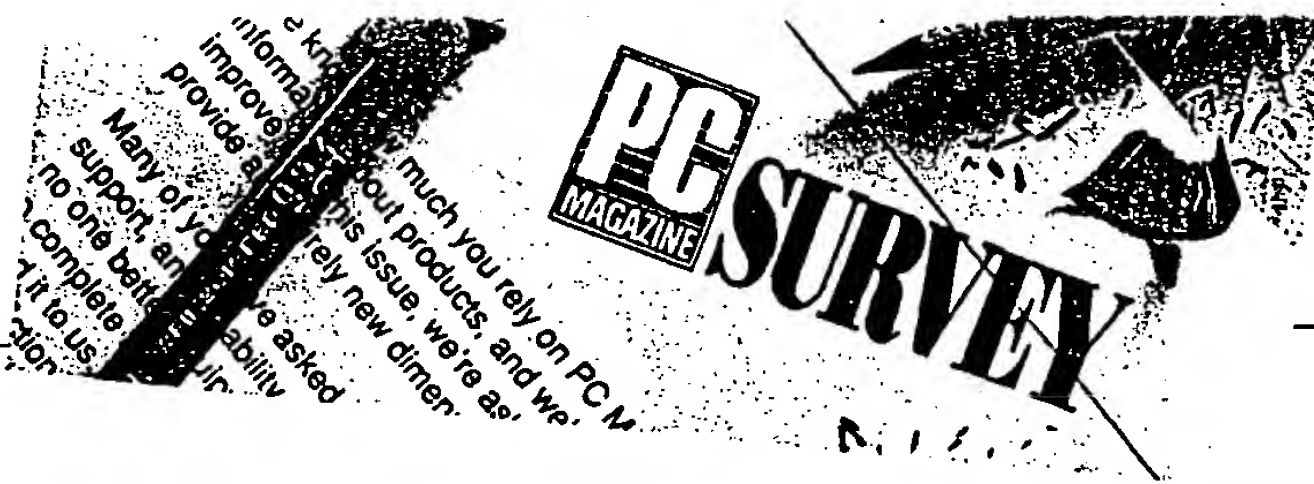
UP FRONT

- 4 INSIDE**
- 15 LETTERS**
- 27 ADVISOR**
Lori Grunin/ DMA and the PC; 386 upgrade boards for 286-based PS/2s; a parallel-port-based LAN adapter.
- 33 FIRST LOOKS**
Zenographic's SuperPrint speeds up *Windows* printing.
DR DOS 5.0 is Digital Research Inc.'s latest and best version of its charismatic DOS clone.
FastCAD 3-D combines 3-D modeling with the magic of *PhotoRealistic Renderman*.
Word Perfect Office: Desktop utilities and application integration for standalone or network PCs.
- 53 NEW AND IMPROVED**
Rink Murray
- 61 READ ONLY**
Lori Grunin
- 63 PIPELINE**
Gus Venditto
- 69 BILL MACHRONE**
Why Johnny Can't Compute
- 75 JOHN C. DVORAK**
Trade Shows and the Future
- 77 Inside Track**
- 81 JIM SEYMOUR**
900-Numbers: Where Support Is Headed?
- 89 WILLIAM F. ZACHMANN**
Microsoft's "Window" to OS/2



COVER STORY

- 99 Service, Support, and Reliability: Agenda for the Nineties**
Bill Machrone/ Not even PC Labs can quantify service, support, and reliability. So we consulted the best information base in the world: *PC Magazine* readers. Almost 20,000 of you filled out our survey, rating the equipment and services you have received from more than 70 companies.
 - 100 Head to Head: Products Compete on Service and Reliability**
 - 104 A Readers' Guide to This Issue**
Mary Kathleen Flynn and Paul B. Ross/ A mini-Michelin guides you through the survey results, score boxes, charts, and reviews in this special issue.
 - 105 The Score Boxes, Line by Line**
 - 111 Desktop PCs: The Long Road to Reliability**
Mary Kathleen Flynn/ Nearly 18,000 readers have rated the service and support they got from 27 system vendors. But numbers tell only part of the story. We phoned hundreds of survey participants to hear their tales of competence and incompetence, frustration and satisfaction, irritation and joy.
- | | | |
|--|---|--|
| <ul style="list-style-type: none"> 114 Acer America Corp. 115 Advanced Logic Research 121 AST Research Inc. 121 AT&T Data Systems 122 Austin Computer Systems 124 Commodore 127 Compaq Computer Corp. 138 CompuAdd Corp. | <ul style="list-style-type: none"> 141 Dell Computer Corp. 142 DTK Computer Inc. 146 Epson America Inc. 147 Everex Systems Inc. 150 Gateway 2000 154 Hewlett-Packard Co. 158 Hyundai Electronics 158 IBM Corp. 164 Kaypro Corp. 166 Leading Edge Inc. | <ul style="list-style-type: none"> 168 NEC Technologies Inc. 168 Northgate Computer Systems 175 Packard Bell 178 PC Brand Inc. 180 Swan Technologies 183 Tandy Corp. 186 Wyse Technology Inc. 188 Zenith Data Systems 189 Zeos International |
|--|---|--|
- 197 Monitors: In the Eye of the Beholder**
Michael Cohn/ Readers take an unflinching look at monitors from 22 manufacturers and tell us which ones kept shining and which blinked back.
 - 258 Service and Support Options**
A comparative table covering service and support policies of all the vendors in this issue.










COVER STORY

- 287 Graphics Adapters: Mostly Trustworthy**
Rock Miller/ Graphics adapters don't fail very often, but when they do, you'll want the manufacturer to back you up with knowledgeable tech support, timely driver updates, and prompt repair service. *PC Magazine* reviews the performance of 11 companies in providing service and support.
- 300 Graphics Adapters: Incidence of Problems**
- 329 Hard Disks: Drive Carefully**
Donald P. Willmott/ Despite the fragility of their moving parts, our survey of 13 hard disk manufacturers shows that hard disks are reliable—more reliable, in fact, than the service some users have received.
- 334 Conquering Disk Boot Failures**
- 369 Printers: Making Good Marks**
Tom Unger/ Frustrating as printer breakdowns may be, at least they tend to be few. And although printers exhibit a host of minor problems, on the whole they turn out to be models of reliability.
- 374 When Good Printers Go Bad:
A Rogue's Gallery of Printer Problems**
- 405 Portables: Long-Distance Runners?**
Jennifer Zaino/ Survey respondents expect their portables to be well equipped to handle the demands of life on the road. Find out how well a dozen top vendors meet those product and service expectations.
- 429 Third-Party Service: From Warranty to Eternity**
Christopher Stetson/ What do you do when your warranty expires? Some users turn to national or regional third-party service providers.
- 431 Reading the Fine Print: Anatomy of a Warranty**

AFTER HOURS

- 537** ■ That old research standby, the encyclopedia, has made the transition to CD-ROM. We examine three familiar titles in the new format.
- 539** ■ *The Game of Harmony*, an arcade game, stresses logic and ingenuity instead of mass carnage.
- 539** ■ *Balance of the Planet* challenges gamers to come down to earth and manage our failing environment.
- 544 ABORT, RETRY, FAIL?**
Bill Howard

PRODUCTIVITY

- 443 LAB NOTES**
Facing the Truth About 16-bit VGA Display Adapters *Alfred Poor*
- 447**  **UTILITIES**
CMDEDIT: The PC Magazine DOS Command-Line Editor
Ashok P. Nadkarni
- 459**  **ENVIRONMENTS**
Metafile Support Under the OS/2 Graphics Programming Interface *Charles Petzold*
- 467 WINDOWS**
William S. Hall/ Here are a few ways to create multiple configurations and some simple methods that will delete the logo and the executive in *Windows 2.0*.
- 471**  **DATABASES**
Trudy Neuhaus/ A solution to the task of left-padding with zeros; transform numeric expressions into their English equivalents.
- 477**  **SPREADSHEETS**
Craig Stinson/ @D360 in earlier versions of 1-2-3; blanks versus labels in 1-2-3 functions.
- 481**  **THE WORKING WORD**
Craig L. Stark/ Use *WordPerfect 5.0* to download your laser printer fonts more quickly.
- 483**  **USER-TO-USER**
Neil J. Rubenking/ Simplify formatting with DOS 4.x switches; locking your computer.
- 485**  **TUTOR**
Jeff Prosisel/ Understanding *PostScript*—a page description language that makes it simple to generate complex images on a printer.
- 489 CONNECTIVITY**
Frank J. Derfler, Jr./ When to use unshielded twisted-pair wires; how cellular telephones can connect mobile PCs to the office.
- 499** Direct Marketing Connection
- 503** Reader Service Card
- 518** Marketplace
- 535** Editorial Product Index
- 536** Coming Up
- 540** Advertisers' Product Index
- 543** Index to Advertisers



Editor-in-Chief and Publishing Director Bill Machrone

Executive Editors Bill Howard, Gus Venditto

Workgroup Systems Editor Frank J. Derfler, Jr.

Senior Editors Trudy Neuhaus (Productivity), Robin Raskin (First Looks)

Managing Editors Diane D'Angelo, Paul B. Ross

Manager, Copy Edit Karen A. Carter Associate Editors Julie Cohen (PCs), Mary Kathleen Flynn (Hardware), Jonathan Matzkin (After Hours), Rock Miller (Graphics), Donald P. Willmott (Software)
Assistant Production Editors Jeanne Albrecht, Nina Barringer, Alan Cohen Staff Editors Glen Becker, Michael Cohn, Carol Levin, Matthew J. Ross, Ann Sherman, Craig L. Stark, Sharon Terdeman, Tom Unger, Jennifer Zaino Assistant Editors Karen Best, Stephanie Izarek, Ellyn McCasland, Michael W. Muchmore, Rink Murray, Ann Ovodow, Oliver Rist, Kim Schueler, Shari L. Serio Editorial Assistants Robin B. Bornstein, Cristina Cordova, William P. Flanagan, Wendy Murch, Emerson A. Torgan, Nathaniel Zelnick Proofreader Gavin Edwards General Business Manager Tom McGrade Associate Business Manager Allen Wollman Network Manager Joe F. Rizzo Librarian Thomas W. Giebel Assistant to the Publishing Director David Baker Assistant to the Executive Editors Christina M. Evelyn Administrative Assistants Christina Okang, Freida T. Smailwood Database Assistant Dolores Williams Interns Meredith Miranda Gadsby, Susan G. Thomas

Contributing Editors Frank Bican, Douglas Boling, Bruce Brown, Ray Duncan, John C. Dvorak, Mitt Jones, Stephen Manes, Michael J. Mefford, Edward Mendelson, Bill O'Brien, Charles Petzold, Alfred Poor, Jeff Prosser, Winn L. Rosch, Neil J. Rubenking, Jim Seymour, Richard Hale Shaw, Barry Simon, Luisa Simone, Craig Stinson, M. David Stone, Ethan Winer, William F. Zachmann

Manager, PC MagNet Christopher Barr Sysops, PC MagNet Rick Ayre, Jesse Liberty Intern Peter Kozodoy

Director, Design and Electronic Publishing Gerard Kunkel

Art Director Jerry Tortorella Associate Art Directors Charles Conover, Patricia Isaza, Mariano Nicieza, Robert C. Smith Associate Graphics Director Gary W. Kaplow Assistant Art Director Ilene Gross Electronic Graphic Artists Simone Blank, David Foster, Michael L. Graham Intern Shadrach Todd

Manager, PC Labs Bruce Freeman Project Leaders Lori Grunin (Graphics), Robert W. Kane (Hardware), Wendy Dugas Pérez (PCs), Abe Rosner (Software), Randol Tigrett (Workgroup Systems) Senior Programmer Stuart R. Greenberg System Support Manager Charles Rodriguez System Support Specialists Mario Baldasserini, Stephen W. Plain Lab Assistants Laura Cox, Carolyn Falconer, Margaret A. Piemonte, Lisa Reittinger Inventory Control Manager John R. Delaney Inventory Control Coordinators Peter Bastide, John Jones Secretary Ann Jo Sanchez

Publisher Ronni Sonnenberg

Director, Marketing Ellen Atkinson Public Relations Jessica Kersey Events Stacy O'Connell Market Information Michael Van Engen Sales Development Lee Uniacke Promotion Kurt Flamer-Caldera Subscriptions Charles Mast Single Copy Sales James E. Gerth

Assistant Production Director John Ansaldi Advertising Production Managers Michael Genzale, Vicki Egan Root Advertising Production Assistant Maryann Pritchett Editorial Production Coordinators Kevin S. Fagan, Jo-Ann Hirschel Production Systems Supervisor Linda Harms Production Systems Assistant Lillian Gaffney

Advertising Office: One Park Ave., New York, NY 10016; (212) 503-5100

ZIFF-DAVIS PUBLISHING COMPANY

President William B. Ziff, Jr.
Executive Vice President Eric Hippeau
Senior Vice President, Marketing Paul H. Chook
Group Vice President J. Scott Briggs
Group Vice President J. Samuel Huey
Vice President, Operations Baird Davis
Vice President, Business Manager T.L. Thompson
Vice President, Controller John Vlachos
Vice President, Circulation Bert Lacy
Vice President, Circulation Services James Ramaley
Vice President, Creative Services Herbert Stern
Vice President, Marketing Services Ann Pollak Adelman
Vice President, Production Roger Herrmann
Vice President, Research Marian O. White
Vice President James W. Stafford
Director, Classified Sales Paul Stafford
Director of Planning Gary A. Gustafson
Production Director Walter J. Terlecki

ZIFF COMMUNICATIONS COMPANY

Chairman William B. Ziff, Jr.
Vice Chairman Philip B. Korsant
President Kenneth H. Koppel
Executive Vice President Philip Sine
Senior Vice President Hugh Tietjen
Vice President, General Counsel, and Secretary J. Malcolm Morris
Vice President, Controller, and Treasurer Patrick J. Burke, Jr.
Vice Presidents Seth R. Alpert, Steven C. Feinman, William L. Philips

HOW TO CONTACT THE EDITORS

The editors of *PC Magazine* want to hear from you. Please send your questions, complaints, compliments, and submissions to *PC Magazine*, One Park Ave., New York, NY 10016. Send electronic mail to MCI Mailbox 157-9301 (*PC Magazine*) or PC MagNet (see below). *PC Magazine's* general number is (212) 503-5255. The West Coast Operations number is (415) 378-5500. For other Ziff-Davis publications, call (212) 503-3500.

We are unable to look up stories from past issues, recommend products, or diagnose problems with your PC by phone. If you bought a product advertised in *PC Magazine*, are dissatisfied, and can't resolve the problem, write (do not call) Lauren Schultz, Advertising Department, at One Park Ave. Include copies of correspondence.

PC Magazine is under no obligation to review unsolicited products. All submissions accepted for publication become the property of *PC Magazine*. We will pay \$50 for Productivity user tips and Abort, Retry, Fail? submissions. For submissions to Abort, Retry, Fail?, if you send artwork for possible reproduction, mark the funny part on a photocopy, not the original.



PC MAGNET: PC MAGAZINE'S ON-LINE SERVICE

PC Magazine operates PC MagNet, a 24-hour-a-day interactive on-line service where readers can find the utilities published in every issue, download an index of all product reviews, share opinions with the editors, and gain access to Computer Library. To join, set your communications software for either 300 or 1,200 bits per second, 7 data bits, even parity, 1 stop bit, and full duplex. To find the number nearest you, dial (800) 346-3247. When the modem connects, press <Enter>. At the HOST NAME prompt, enter PHONES. Follow the menus and note the number closest to you.

After you have found the phone number, hang up and dial using the number you just found. When you connect with PC MagNet, press Ctrl-C. At the HOST NAME prompt, enter CIS. At the USER ID prompt, enter 177000,5000. Enter PC*MAGNET at the PASSWORD prompt and Z10D9000 at the ENTER AGREEMENT NUMBER prompt. CompuServe members can join by entering GO PCMAG at any CompuServe prompt. For Customer Service, call (800) 848-8990. PC MagNet costs \$12.80 per hour via MasterCard, VISA, or American Express.

PERMISSIONS, REPRINTS

Material in this publication may not be reproduced in any form without permission. If you want to quote from an article or use *PC Magazine's* logo in conjunction with an Editor's Choice designation, write Chantal Lavelanet; for information on reprints, write Jennifer Locke.

SUBSCRIPTION QUESTIONS

If you want to subscribe to *PC Magazine*, if you have a question about your subscription, or if you're moving, call or write: *PC Magazine*, P.O. Box 54093, Boulder, CO 80322; (800) 289-0429, (303) 447-9330 (outside the U.S.). A 1-year subscription (22 issues) costs \$44.97. Canada and foreign: add \$31.00 per year. New subscriptions and address changes take 6 to 8 weeks to process. For back issues, write Kim Armstrong, Ziff-Davis Publishing Company, One Park Ave., 5th floor, New York, NY 10016. Send \$8 per issue (check or money order); \$9 outside the U.S.

PC MAGAZINE REVIEWS

Except where noted: All *PC Magazine* reviews are of currently available products. Comparative reviews are arranged alphabetically by product name for software, by company name for hardware. Software is free from copy protection. On benchmark test charts, shorter bars indicate faster products (less time to complete a task). Benchmark and performance tests are run on 8-MHz IBM PC ATs.



Editorial and Business Office: One Park Ave., New York, NY 10016. Editorial: (212) 503-5255; Advertising: (212) 503-5100. *PC Magazine* is an independent journal, not affiliated in any way with International Business Machines Corp. IBM is a registered trademark of International Business Machines Corp. Entire contents copyright ©1990 Ziff-Davis Publishing Co., a division of Ziff Communications Co. All rights reserved; reproduction in whole or in part without permission is prohibited. The following are registered trademarks of Ziff Communications Co.: PC, PC Magazine, PC Labs, PC MagNet, PC Magazine Award for Technical Excellence, PC Magazine Editor's Choice Award. The following are trademarks of Ziff Communications Co.: Abort, Retry, Fail?; Connectivity; Databases; Direct Marketing Connection; First Looks; New and Improved; Advisor; Lab Notes; PC Marketplace; Pipeline; Tutor; The PC Utilities; Power Programming; Power User; Read Only; Spreadsheets; User-to-User; The Working Word.

edited by
Trudy Neuhaus

Databases

DATE CONVERSION

This is a user-defined function that converts a date expression in *dBASE IV* into a more readable version. I use this simple function with all of my code; it makes any computer-generated report more personable.


```
FUNCTION cdate
PRIVATE mdate
PARAMETER mdate
RETURN (CMONTH(mdate) + " " +
LTRIM(STR(DAY(mdate), 2)) + " " +
STR(YEAR(mdate), 4))
```

For example,

```
mbirth = CTOD("3/10/61")
? CDATE(mbirth)
* Result: March 10, 1961
```

This also can be expanded to include other information such as the day of the week.

Benjo Dans
Saginaw, Michigan

 Indeed, date functions like this one are a mainstay in most programmers' *dBASE* toolkits. Many variations on this theme are possible. For example, to create a function that also returns the day of the week, replace the RETURN statement above with the following:

```
RETURN (CROW(mdate) + " " +
CMONTH(mdate) + " " +
LTRIM(STR(DAY(mdate), 2)) + " " +
STR(YEAR(mdate), 4))
```

The resulting function works like this:

```
mbirth = CTOD("3/10/61")
? CDATE(mbirth)
* Result: Friday, March 10, 1961
```

Or, if you need a shorter version for a report with space limitations, try

```
RETURN (SUBSTR(CMONTH(mdate), 1, 3) +
" " + LTRIM(STR(DAY(mdate), 2)) + " " +
STR(YEAR(mdate), 4))
```

■ **DATE CONVERSION:**
Here's a UDF that converts a *dBASE* date expression into everyday format.

■ **CONVERTING NUMBERS TO WORDS:** A *FoxBASE* program that converts numeric expressions into their English equivalents.

■ **LEFT-PADDING WITH ZEROS:** A combination of *dBASE* functions lets you left-zero-pad a numeric field.

which works like

```
mbirth = CTOD("3/10/61")
? CDATE(mbirth)
* Result: Mar 10, 1961
```

FoxPro users can use a built-in function called MDY, which returns the date in the first format above. However, the year portion is dependent on the SET CENTURY setting. For example, MDY(mdate) returns "March 10, 61" with SET CENTURY OFF (note that "19" is missing). In addition, CENTURY defaults to OFF.

To ensure a complete four-digit year, you may be tempted to build a UDF that locks in the century setting:

```
FUNCTION mymdy
PARAMETERS mdate
PRIVATE savecen, rdate
savecen = SET("century")
SET CENTURY ON
rdate = MDY(mdate)
SET CENTURY &savecen
RETURN rdate
```

This saves and restores the current century setting while returning a date guaranteed to

have four digits. However, you'd be well advised to resist any such temptation, since Mr. Dans's much shorter CDATE function executes about 15 times faster than mymdy and accomplishes the same thing.

CONVERTING NUMBERS TO WORDS

I'm submitting a *FoxBASE* 2.01 program that converts a numeric amount to its English equivalent—a facility necessary in my invoicing program.

Alfredo Conde
MIA Airport, Philippines



A common application for this technique is printing checks. Mr. Conde's original version worked quite well, but only for amounts up to \$999,999.99. Such a limitation won't do! The N2W procedure in Figure 1 has been changed so that it works with numbers up to \$999,999,999,999.99 (just in case you'd like to place a deposit against the national debt). In tests against a file of 50 random numbers, it worked fine.

To see how it works, try the following example:

```
SET PROC to NWORDS
? N2W(150.99)
```

```
* Result is:
* One Hundred Fifty Dollars and
* 99/100
```

You should note that this routine works in *FoxBASE +*. It does not work in current releases of *FoxPro* (through 1.2).

LEFT-PADDING WITH ZEROS

Recently I faced a perplexing *dBASE* problem. My client wanted a database unloaded in SDF (System Data Format) with all numeric fields zero-padded to the left of the most significant digit. How easy, I thought. Simply define the field as character data and use the STR() function to convert any numeric data to character data. But the

Use Northgate OmniKey/102 Risk Free For 60 Days!



Original Northgate design with function keys on the left!

First keyboard to get back to basics with 12 function keys on the left—the way many users prefer! In fact, readers of *Computer Shopper* voted OmniKey/102 their Best Buy!*

Touch that leads to better typing! The secret? OmniKey's ALPS tactile mechanical key switches let you know each keystroke has registered with a precise "click." No need to slow down to "eye check" the monitor—increases your typing speed with NO EXTRA EFFORT!

Find out for yourself ... order your OmniKey/102 NOW! If you're not 100% pleased, return for full refund—including ground shipping charges!

Backed by the industry's strongest warranty—five full years! If you have any problems due to materials or workmanship, Northgate will promptly repair or replace your OmniKey/102 FREE!

OmniKey/102
Another "Smart Tool
for Business"™

ONLY

\$99.00

FOB
Minneapolis, MN

Look at all these outstanding OmniKey/102 features:

- Compatible with virtually all IBM-type personal computers, including: PC/XT/AT and PS/2 systems
- Interchangeable left CTRL, ALT and CAPS LOCK keys. Keep them as shown or put them in a standard IBM enhanced layout; CAPS LOCK next to "A", ALT next to space bar, CTRL under SHIFT. On the right, ALT and CTRL interchange, too
- Swap the right Backslash and Asterisk keys—it's up to you!
- Twelve F-keys on the left—for quick access to CTRL, ALT, SHIFT combination commands!
- Inverted T-shaped cursor control keypad
- Separate calculator-style numeric keypad with added equals key—great for spreadsheet users!
- Large L-shaped Enter, double size Backspace, two oversized Shift keys—easy to hit ... reduces typing errors!
- LED indicators: shows Scroll, Caps, Number lock status at a glance
- Double injected key caps for long life and durability
- ALPS click/tactile mechanical key switches for a crisp, responsive feel and faster, more accurate typing
- Non-skid design! OmniKey/102 has a heavy steel base for durability. Weighs 4.6 lbs.—won't slide around no matter how fast you type!
- Keys color coded for use in Word Perfect
- FCC Class B Certified

Call for the Dealer Nearest You or Place Your Order Direct

800-526-2446

HOURS: Mon.-Fri.
7 a.m. - 9 p.m.; Sat.
8 a.m. - 4 p.m. Central

Notice to the Hearing Impaired: Northgate now has TDD capability. Dial 800-535-0602. FAX Your Order 612-476-6443. Dealer and distributor pricing available. Call 800-328-5564. Charge it on VISA or MasterCard

**NORTHGATE
COMPUTER
SYSTEMS**

"We hear you!"

1 Northgate Parkway, Eden Prairie, Minnesota 55344

Northgate Computer Systems, Inc. All rights reserved. Northgate, OmniKey and the Big N logo are trademarks of Northgate Computer Systems. All other product brand names are trademarks or registered trademarks of their respective owners. Specifications subject to change without notice. All models subject to availability. *Best Buy for 1988-1989.

Databases

NWORDS.PRG

```

.....
* NWORDS.PRG
* Convert numbers to words
.....
PROCEDURE n2w
PARAMETERS m_number          && pass numeric value
*
* Return "Invalid number" if out of range
*
IF (m_number <= 0) .OR. (m_number > 999999999999.99)
RETURN "Invalid number"
ENDIF
*
* Initialize
*
word = {}
orig = m_number
cents = (m_number - INT(m_number))*100
*
* If less than $1.00, start with "zero"
*
IF INT(m_number) = 0
word = {zero}
ENDIF
*
* Set Up Arrays
*
DIMENSION mone(19), mten(9)
mone(1) = {One}
mone(2) = {Two}
mone(3) = {Three}
mone(4) = {Four}
mone(5) = {Five}
mone(6) = {Six}
mone(7) = {Seven}
mone(8) = {Eight}
mone(9) = {Nine}
mone(10) = {Ten}
mone(11) = {Eleven}
mone(12) = {Twelve}

```

```

mone(13) = {Thirteen}
mone(14) = {Fourteen}
mone(15) = {Fifteen}
mone(16) = {Sixteen}
mone(17) = {Seventeen}
mone(18) = {Eighteen}
mone(19) = {Nineteen}

mten(1) = {Ten}
mten(2) = {Twenty}
mten(3) = {Thirty}
mten(4) = {Forty}
mten(5) = {Fifty}
mten(6) = {Sixty}
mten(7) = {Seventy}
mten(8) = {Eighty}
mten(9) = {Ninety}
*
* Handle billions
*
x = INT(m_number/1000000000)
IF x > 0
DO mkwords WITH x, word
word = word + {Billion}
ENDIF
m_number = m_number - (x * 1000000000)
*
* Handle millions
*
x = INT(m_number/1000000)
IF x > 0
DO mkwords WITH x, word
word = word + {Million}
ENDIF
m_number = m_number - (x * 1000000)
*
* Handle thousands
*
x = INT(m_number/1000)

```


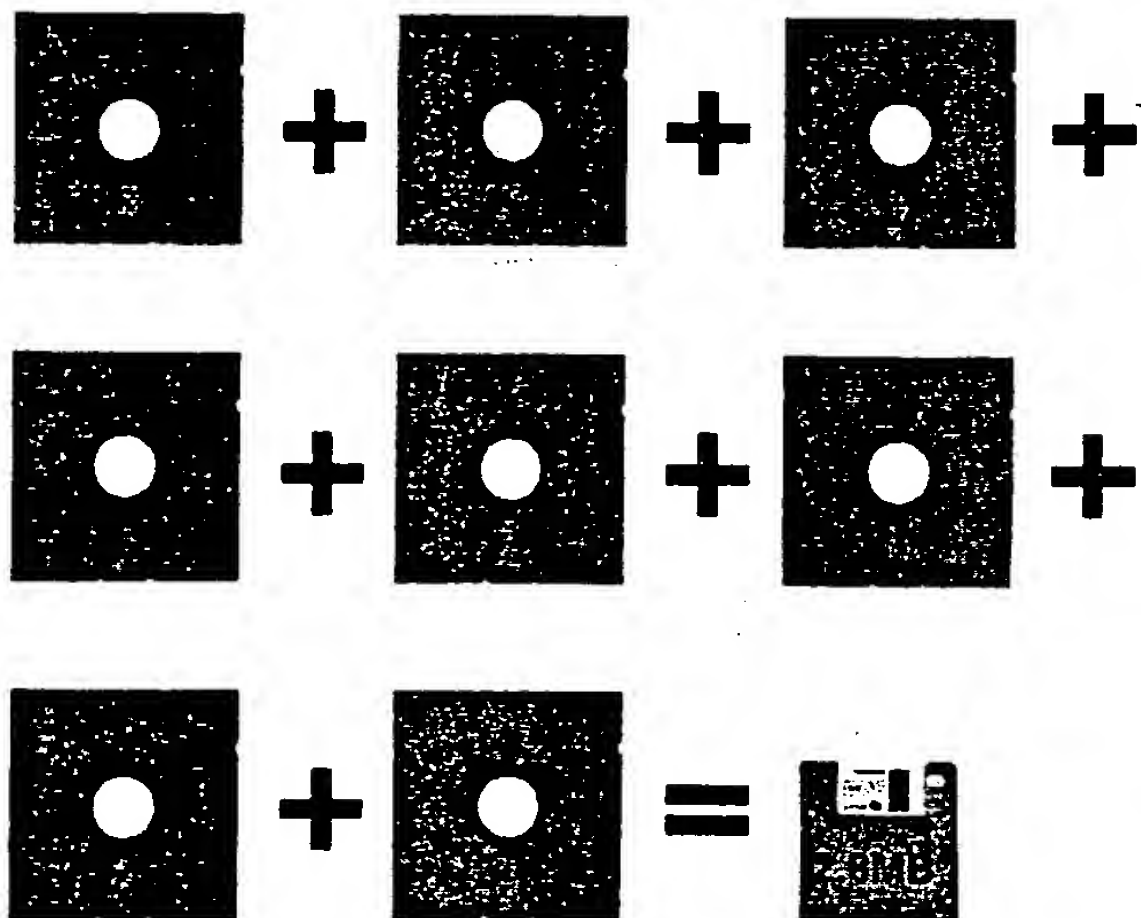
1 of 2


Figure 1: This procedure converts numbers into their English equivalents.



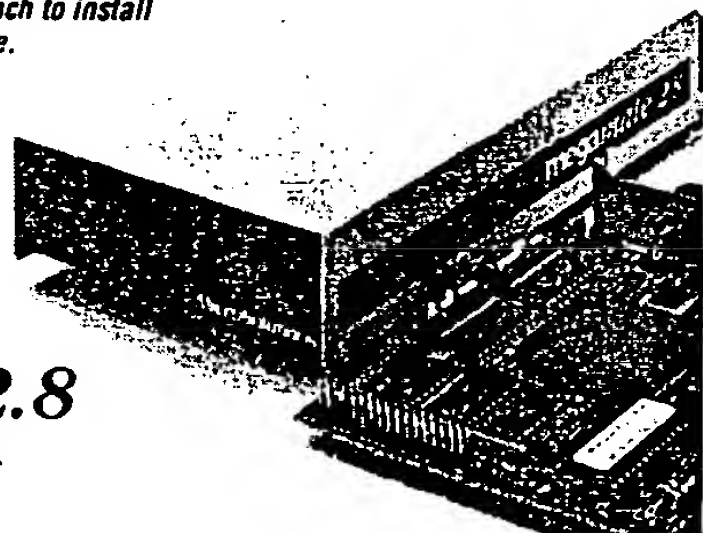
Megamate 2.8 stores eight times as much data as a 5.25" drive at four times the speed. It handles any 3.5" IBM disk - 2.8MB, 1.4MB, 720KB - and is a cinch to install on any IBM PC, XT, AT or compatible.

Megamate 2.8 comes complete with everything you need. So add 2.8MB ED capability the easy way - with Megamate 2.8! Call Micro Solutions today for the dealer nearest you.

1.4MB version also available.

megamate2.8

Micro Solutions 132 W. Lincoln Hwy.
DeKalb, IL 60115 815-756-3411



Play It Again...Sam.

Don't allow your excellent review to fade away like a golden oldie. Ziff-Davis can print custom-designed article reprints* to keep your product at the top of the hit parade. Call today for information: Jennifer Locke, Reprints Manager 212-503-5447.

*minimum quantity 500; 3 to 4 weeks delivery.



Databases

2 of 2

NWORDS.PRG

```

IF x > 0
  DO mkwords WITH x, word
  word = word + [Thousand ]
ENDIF
m_number = m_number - (x * 1000)
* Handle the rest
IF m_number >= 1
  DO mkwords WITH INT(m_number), word
ENDIF
* Setup "dollars" and cents
IF orig >= 2 .OR. int(orig) = 0
  word = word + [Dollars and ]
ELSE
  word = word + [Dollar and ]
ENDIF
IF cents > 0
  IF cents < 10
    word = word + '0' + STR(cents,1) + [/100]
  ELSE
    word = word + STR(cents,2) + [/100]
  ENDIF
ELSE
  word = word + [00/100]
ENDIF
RETURN word
* Procedure: MAKE_WORDS
PROCEDURE mkwords
PARAMETERS numb, string
* PRIVATE nstr, nlen, nlook

```

```

nstr = STR(numb)
DO WHILE SUBSTR(nstr,1,1) = ' '
  nstr = SUBSTR(nstr,2)
ENDDO
DO WHILE ("" <> nstr)
  nlen = LEN(nstr)
  nlook = VAL(SUBSTR(nstr,1,1))
  IF nlook = 0
    IF nlen = 1
      nstr = ''
    ELSE
      nstr = SUBSTR(nstr,2)
    ENDIF
  LOOP
  DO CASE
  CASE nlen = 3
    string = string + mone(nlook)
    string = string + "Hundred "
    nstr = SUBSTR(nstr,2)
  CASE nlen = 2
    IF VAL(SUBSTR(nstr,1,2)) < 20
      string = string + mone(VAL(SUBSTR(nstr,1,2)))
      nstr = ''
    LOOP
  ENDIF
  IF VAL(SUBSTR(nstr,1,2)) >= 20
    string = string + aten(nlook)
    nstr = SUBSTR(nstr,2)
  ENDIF
  CASE nlen = 1
    string = string + mone(nlook)
    nstr = ''
  ENDCASE
ENDDO
RETURN
* eof

```

STR() function doesn't left-zero-pad. So I came up with a technique that combined the STR(), SUBSTR(), and STUFF() functions. The example that follows will left-zero-pad a seven-position field containing the number 257:

```

STORE STR(257,7) to x
STORE 1 to I
DO WHILE I <= 7
  IF SUBSTR(x,i,1) = ' '
    STORE STUFF(x,i,1,'0') to x
  ENDIF
  STORE I+1 to I
ENDDO

```

Terry Blankenship
Brentwood, Tennessee



Numeric formatting is a more common operation required by many business programs. This technique does the job but may be shortened. It can be made to work with any length string by including the SUBSTR comparison as part of the DO WHILE loop:

```

STORE 1 to I
DO WHILE SUBSTR(x,i,1) = ' '
  STORE STUFF(x,i,1,'0') to x
  STORE I+1 to I
ENDDO

```

If your dBASE implementation supports user-defined functions, you can create one to do the converting and padding like this:

```

*
* PROCEDURE ZNUM(number, slen)
*
* This function converts a number
* to a string of length slen left
* padded with leading zeroes.
* Returns the string.
*
PROCEDURE ZNUM
PARAMETERS number, slen
STORE 1 to I
STORE STR(number, slen) to string
DO WHILE SUBSTR(string,i,1) = ' '
  STORE STUFF(string, i, 1, '0') to
  string
  STORE I+1 to I
ENDDO
RETURN string

```

To use ZNUM in your program, you could type

```
string = ZNUM(257, 7)
```

```
* "0000257"
```

Readers will be interested to note that Fox Software's *FoxPro* includes built-in func-

tions called PADL, PADR, and PADC to left-pad, right-pad, and center-pad, respectively. If you're using *FoxBASE+*, here's a handy emulation of *FoxPro*'s PADL function:

```

*
* PROCEDURE PADL(string, length,
* padchar)
*
* This UDF left pads a string to
* length characters padding with
* character padchar. It is similar
* to FoxPro's PADL in that it
* truncates the string if the
* string is longer than length, and
* does not trim leading or trailing
* blanks on the incoming string.
*
PROCEDURE PADL
PARAMETERS string, length, padchar
IF LEN(string) > length
  STORE SUBSTR(string,1,length) TO
  string
ENDIF
DO WHILE LEN(string) < length
  STORE padchar + string TO string
ENDDO
RETURN string

```

The PADL function accepts three arguments: the string to pad, the desired length,

Databases

and the character to use for padding. You could use it in your programs like this

```
string1 = PADL("257",7,"0")
```

```
* string1 = "0000257"
```

```
string2 = PADL("257",7,"")
```

```
* string2 = "*****257"
```

If you're using *dBASE IV* or *FoxPro*, another alternative is the **TRANSFORM** function. This function lets you apply PIC-

FoxPro includes built-in functions that enable users to left-pad, right-pad, and center pad.

TURE formatting to character, logical, date, and numeric data and returns the resulting string.

The function codes for leading zeros and right justification are L and R respectively, so the following does the trick:

```
string1 = TRANSFORM(257,"@LR  
9999999")
```

```
* string1 = "0000257"
```

LET US HEAR FROM YOU

Share your database experience, tips, and techniques and we'll pay you \$50 for any submissions we print. Submissions must be made on-disk; be sure to include a print-out, too. Mail your contributions to Databases, *PC Magazine*, One Park Avenue, New York, NY 10016, or upload them to PC MagNet (see the "By Modem" sidebar in the Utilities column for instructions.)

Research for this article was provided by Salvatore Ricciardi, a New York-based consultant and frequent contributor to *PC Magazine*.

Q: What Do All These Screens Have In Common? A: They Were Created Using PathMinder+

With *PathMinder+*'s Reconfigurable User Interface (RUI) and a mouse, you can easily create an interface for anyone from a secretary to the CEO.

The power of RUI is limitless. If you have a novice user and you don't want them to erase files, no problem. Just remove the erase command from the menus and your worries are gone. If another person knows how to use their word processor and doesn't have time to learn to use *PathMinder+*, you can make *PathMinder+*'s keys mimic their word processor. Best of all you can customize each user's interface without any programming. With a few keystrokes or clicks of the mouse you can layout an entire user environment.

Unlike the competition, *PathMinder+* gives you a complete communications package by Crosstalk, not just E-mail capability. *PathMinder+* is the disk management system that can provide all this power to everyone.

With *PathMinder+* you can tailor the program to fit the user's needs and experience level. When it comes to managing data, you'll find *PathMinder+* is the easiest way... and the best way. So, go ahead and give *PathMinder+* a try.

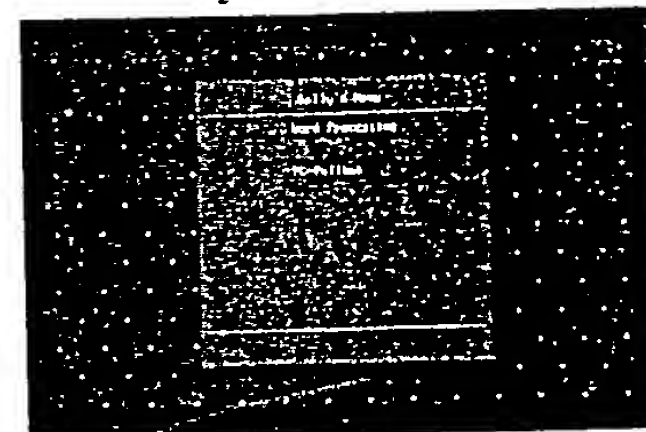
- ✓ Reconfigurable User Interface (RUI)
- ✓ Application Management
- ✓ File Management
- ✓ Directory Management
- ✓ Data FileViewers
- ✓ System Log
- ✓ Mouse Support
- ✓ Communications
- ✓ Text Editor
- ✓ and MORE

Finalist

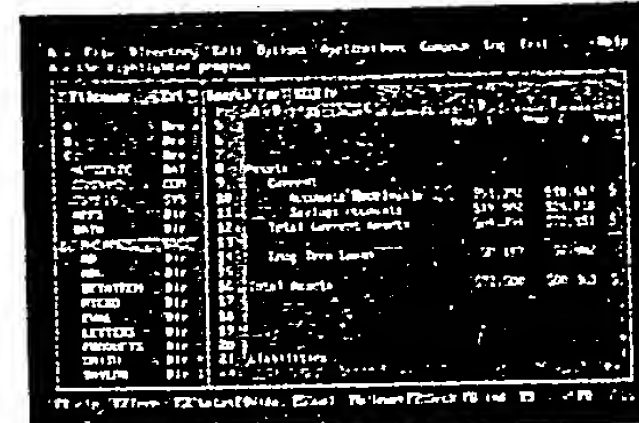


Optimized for 286/386's

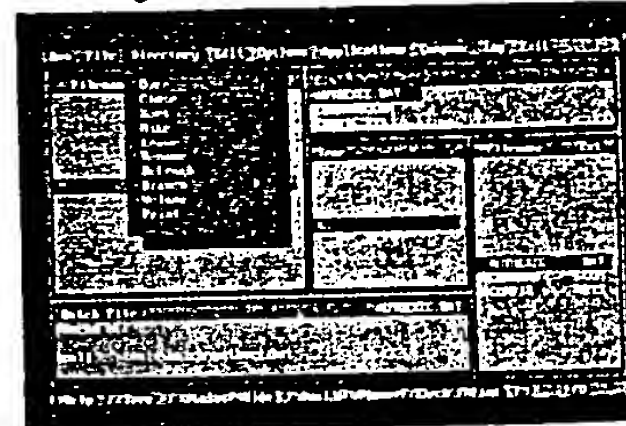
The Secretary



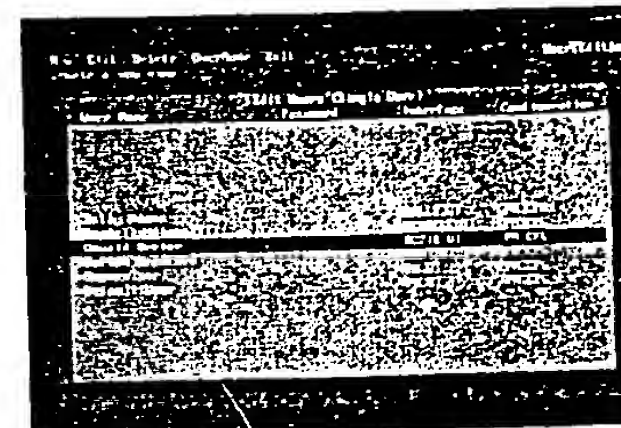
The Accountant



The Engineer



Fill In Your Name Here



Only \$149.⁰⁰
Suggested Retail

**WESTLAKE
DATA CORP**

(512) 328-1041, FAX (512) 328-1040, P.O. Box 1711, Austin, TX 78767

PathMinder+ is a trademark of Westlake Data. All other products are trademarks or registered trademarks of their manufacturers.

Volume 10, Number 2
June 1986

42

MIS Quarterly

Management Information Systems

MATH SCIENCES

JUN 9 1986

LIBRARY



Sponsored by: The Society for Information Management and
The Management Information Systems Research Center

Issues and Opinions

What Will the Change of the Millenium Do to Our Data Processing Systems?

The arrival of the next century, the year 2000, has implications which must be planned for by data processing professionals. Current computerized information systems store dates in Gregorian (YY/DDD), Julian (MM/DD/YY) or standard (YYMMDD) formats; these formats use no more than two digits to describe the year. Two digits are not enough for an orderly representation of dates at the end of the century with respect to dates at the beginning of the 21st century.

To make the matter clear, consider the following example. In medical insurance claims processing, the system checks each claim to see if the policy effective date is before the date of medical service for which coverage is claimed. Suppose the policy effective date is April 1, 1997, and medical service was rendered after the effective date, namely on April 1st, 2002. The traditional system would compare the dates represented in standard format. The test would incorrectly reject the claim since 970401 is greater than 020401. This is a simple example and the knowledgeable reader will probably be able to construct many other examples.

The problem lies with the fact that two significant digits to describe the year will not be enough. Furthermore, since at the end of this century the millenium changes too, three digits are not enough either.

The necessity of changing the date format will be felt both in the areas of system design and in maintenance. As the end of the century approaches, systems analysts will become aware of the fact that the life expectancy of a new system being designed will go past the end of the century, and the problem will be tackled in the analysis phase.

In other cases where no special provisions will have been made for handling the new date format, maintenance projects will be undertaken. The size of a maintenance project to change the date format of a system is not a negligible one. A simple example is offered by the zip-code change from five to nine digits. Although no time pressure for the change has been applied in most cases, the change requires a considerable amount of maintenance effort. Online programs, batch programs and data stores have to be converted. The elements of the system which are affected by the change have exceeded the number of those which actually process the zip-code information. Programs and data stores had to be converted just for the sake of system consistency. In the case of the date format change, the aggregate impact on the data processing industry will definitely be large since this change will have a deadline.

Systems to undergo date format changes will be primarily systems which will, at that time, function beyond their initial projected lifetime. It is precisely the systems which are designed today that are most likely to face this problem. It is the systems we design today which will have exceeded their lifetime and for which organizations will find it more difficult to justify maintenance work.

The high personnel turnover in the DP industry is a factor which indirectly contributes to the deferment of the date format change problem. Most of the individuals who are likely to be responsible for ensuring proper functionality of the DP systems at the turn of the century will probably face the problem in a different position or place than they are now. The result: too little attention is currently being paid to the issue in question.

The following are recommended steps in the analysis process by which necessary date format changes are identified and solutions developed:

1. Identify the date comparisons.
2. Identify the sort procedures which involve date keys.
3. Identify whether date comparisons and sorts are performed for dates across different periods like months or years.
4. Estimate how long the problem will persist

if no change is undertaken and evaluate the cost of making no system modification.

5. Design the program and data store changes.
6. Develop a conversion and implementation plan.

The system changing needs will vary widely from one system to another. As two extreme cases, an insurance system will typically need many modifications, while in an accounting system the problem can be eventually solved by judiciously handling the closing periods.

Systems which are currently being designed should explore the impact of the millenium change. As the end of the century approaches, each organization should at least initiate a major analysis project to estimate and evaluate the impact of the date change and decide upon one of the following four alternatives:

1. Accelerate the termination of the old system and replace it with a new system which is able to handle the date transition.
2. Modify the existing system, in which case design, planning and conversion has to take place. If data structures are being changed as a result of the modification requirements, then the project will be of considerable size and adequate use of database dictionaries can help the conversion process.

3. Modify the system locally. In this case, careful analysis must be conducted to identify the boundaries of the change.
4. Do nothing.

There will be a great variability of the degree of impact from one system to another and from one organization to another. No automatic conversion will be possible and skilled technical professionals will be needed to carry out the changes. It is even possible that the date format change will have an impact on the job market, creating an excess demand for qualified programmers and possibly an excess supply of such individuals when the crisis is over. The date change problem at the end of the millenium should not be underestimated and the time has come for analysts and DP managers to think about it.

Dan R. Pieptea
The University of Texas at Dallas
Richardson, Texas

0000-0000
0000
v. 9
no. 11
November
1994

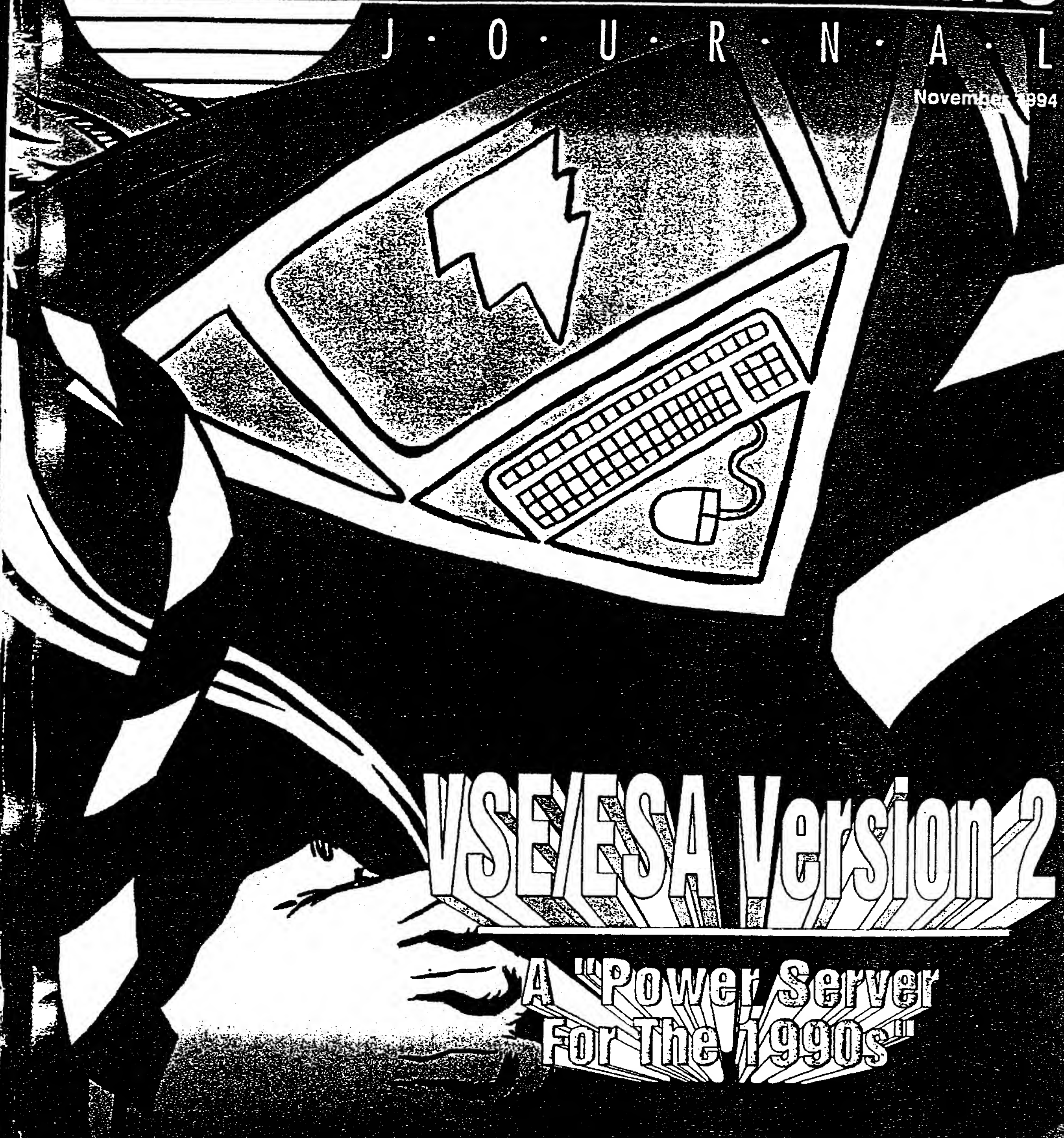
Main Ser
1052-6566
Received on: 11-17-94
Enterprise systems journal

IBM Host-Based Enterprise-Wide Computing

ENTERPRISE SYSTEMS

J · O · U · R · N · A · L

November 1994



VSE/ESA Version 2

A "Power Server
For The 1990s"



I/S Management

- 12 20 Questions For Gottfried Santschi Of ELVIA Life
By John Kador
- 18 New Technologies For Mission-Critical Applications
By Jnan R. Dash

Operating System Platforms

- 22 VSE Central: VSE/ESA Version 2 — A "Power Server For The 1990s"
By Eric L. Vaughan
- 48 Performance Assurance: A New Paradigm For Performance And Capacity Planning
By G. Jay Lipovich

Client/Server Enterprise Solutions

- 30 Client/Server And Legacy Systems Integration: Integrating The Old With The New
By Travis Richardson
- 68 Rightsizing And Client/Server: Notes From The Field
By Bill Gourgey
- 74 Measuring End-To-End Availability
By Mike Bonett

Database Management

- 38 Exploiting DB2 2.3 Performance Features
By Lee Prince

Data Center Automation

- 58 Networking Around The World: Automated Network Management Provides Business Solutions
By Arnold Farber and Rosemary LaChance

Contingency Planning

- 83 Business Resumption Planning: An ROI Opportunity
By Paul B. Myatt
- 89 Performing Recoveries In An Open Systems Environment
By Craig Knutson and Anouar Jamoussi



IBM's recent announcement of VSE/ESA Version 2 positions this operating system firmly in the open, client/server computing environment. IBM calls this new version the "power server for the 1990s." In his article on page 22, Eric L. Vaughan presents a detailed look at the new VSE and how it fits this new designation.

Cover illustration by John H. Foote

First Impression

- 8 A-Net™ Offload (Teubner & Associates)
- 10 DateServer™ 2000 (Computer Software Corp.)

Departments

- 4 Publisher
- 8 Inside IBM
- 80 Advertiser Index
- 94 Humor
- 98 Viewpoint

ESJ Supplement

Distributed Computing Solutions: Utilizing Today's Technologies For Tomorrow's Competitive Advantage

DateServer™ 2000

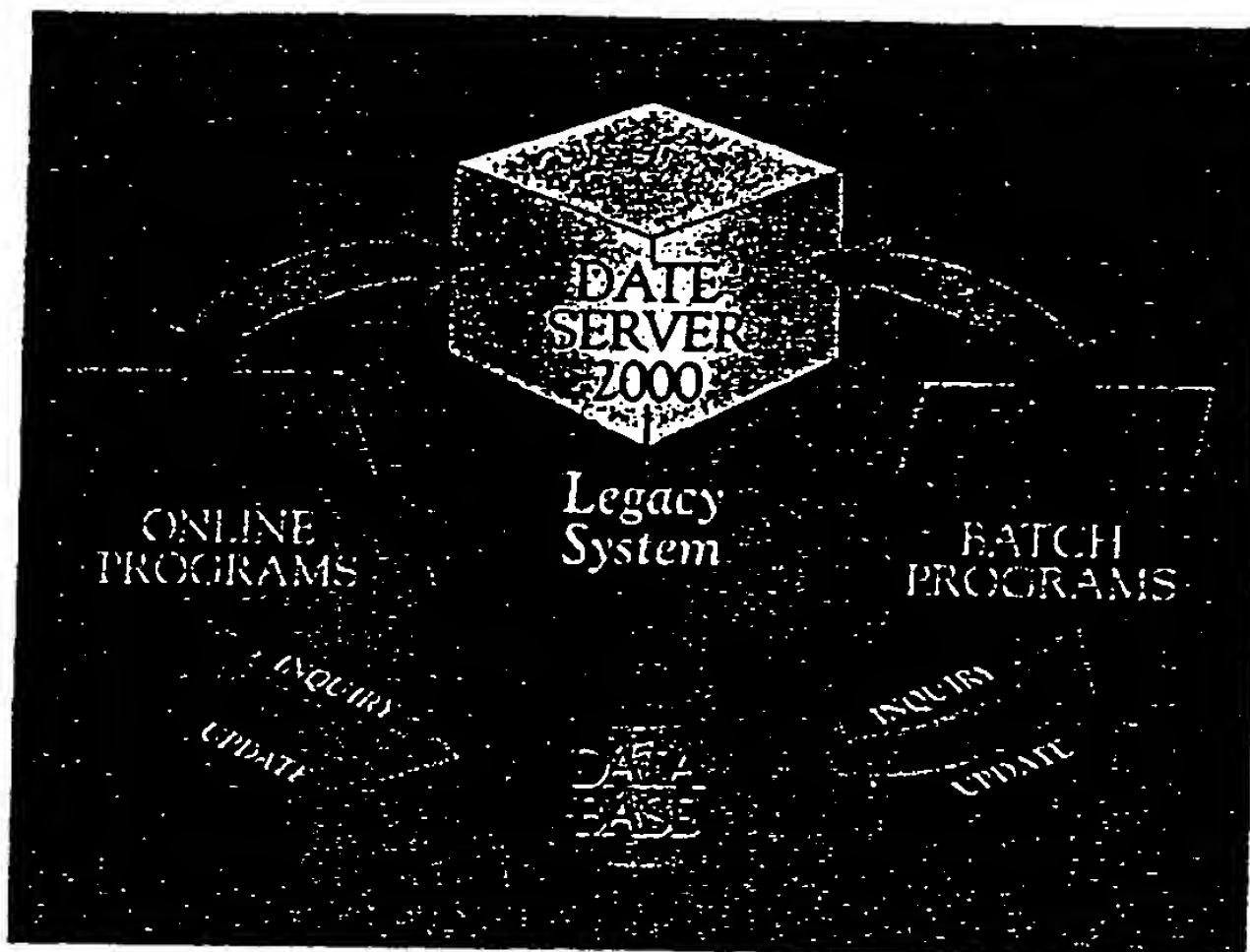
Computer Software Corp. Prepares Legacy Systems For Year 2000

For many organizations, preparing legacy systems to correctly process multcentury dates will likely require a major expenditure of time and money over the next few years. Changes must be made to all application systems that make date field comparisons or calculations using date field values with only a two-digit representation for the year. Computer Software Corp.'s DateServer 2000 software offers a unique solution to significantly reduce the time and effort to prepare for processing in the year 2000 and beyond.

The conventional approach to the year 2000 problem is to expand date field sizes to provide for date values with a four-digit year. This file-conversion, or "sledgehammer," approach is a costly solution for most enterprises. It requires special programs to convert all date fields within all databases. Changes to all existing input screens and files are also required. It requires coding changes to all current programs that process the new input dates and the converted databases as well as changes to all output screens and reports. Finally, this conventional approach requires the testing of all the new and changed programs, and a great deal of coordination during the actual system conversion and implementation.

No Database Conversions

In contrast, the DateServer 2000 solution eliminates the requirement to convert any files or databases. Using the familiar CALL statement interface, pro-



grammers can access DateServer 2000 routines to make date field comparisons or calculations on all date fields in their present database formats. During program execution, DateServer 2000 routines will assign the appropriate century value, based on the system's current date and DateServer 2000 installation parameters, to correctly perform requested date comparisons or calculations.

One Program At A Time

Eliminating the need for file or database conversions and the requirement to expand input or output date formats, the DateServer 2000 solution simplifies the preparation and coordination efforts required to process multcentury dates. Application programs may be changed, tested and implemented one program at a time without worry about complex database interaction with other on-line or batch programs. Rather than having to fit a major database conversion into a busy project schedule, these appli-

cation program changes could be merged with other project testing.

DateServer 2000 software provides a comprehensive, efficient and convenient set of routines for date comparison, calculation and/or manipulation. The routines support 36 date formats including character, packed and binary Gregorian and Julian dates, and *day of week* and *relative day* values for all dates in the Gregorian calendar. The routines perform: date validation; date comparison; conversion from one format to another; determina-

tion of *day of week* and *relative day* values; calculation of the difference between any two dates in number of years and/or days; and determination of the date resulting from the addition or subtraction of any number of days to or from a given date.

Even enterprises that choose the file conversion approach for some legacy systems could still use the DateServer 2000 solution for other systems.

Computer Software Corp. offers versions of the DateServer 2000 software to operate in either an IBM MVS or IBM VSE environment. A CICS demonstration program is also included for CICS users. Priced on a tiered basis with multiple-site discounts, DateServer 2000 software is available starting at \$10,000. For more information, contact Computer Software Corp., 19100 Detroit Road, Cleveland, OH 44116, (800) 908-2000, Fax (216) 333-8288, or (216) 333-9080 outside the United States. ☉

— Chris Roberts

CIRCLE #74 on Reader Service Card ▲

Charmar Enterprises. Inc.

THE CHARMAR CORRECTION

by

William H. Schoen

Copyright 1984.
All rights reserved.

Charmar Enterprises. Inc.

CONTENTS:

	PAGE
1. THE YEAR 2000 PROBLEM: AN EXPLANATION.	3
2. ITS MAGNITUDE.	4
3. WHY A SOFTWARE FIX IS HIGHLY UNLIKELY.	5
4. WHY NOTHING IS BEING DONE.	6
5. THE TIME TO ACT: NOW	7
6. THE FOUR-DIGIT YEAR: PART OF THE SOLUTION.	8
7. THE CHARMAR CORRECTION:	
A. THE BASICS.	9
B. RECOMMENDED POLICY.	10
C. THE TWO-DIGIT YEAR DIRECTIVE.	11
8. THE 'YR2000' SUBROUTINE.	12-21
9. THE 'FDYEAR' SUBROUTINE.	22-24
10. LINK INSTRUCTIONS.	25
11. EXHIBITS: A,B,C,D,E.	26-30

Charmar Enterprises. Inc.

THE YEAR 2000 PROBLEM:
THE SERIOUS PROBLEM
IGNORED BY THE ENTIRE DATA PROCESSING COMMUNITY!

As this is being typed, the date is:12/14/83.

16 years, 17 days later, the date will be:12/31/99.

The following day's date will be:01/01/00.

Take another look at that date -- 01/01/00 -- doesn't it look a bit odd? Exactly what year is it? Is it the year 2000, the year 1900, or some other year?

"By that time", You might say, "I'll just assume it means the year 2000."

Congratulations. You're smarter than any computer that's ever been made.

Unfortunately, almost all of the software written up to now in COBOL, PL/I, and most other languages has neglected to include logic to process 21st Century dates properly. If nothing is done about this, when the year 2000 arrives, you will be unable to enter it into your systems so that your system will recognize that it is greater than the year 1999, and a great deal of your software either won't run or will produce unpredictable results.

This predicament is due to the fact that the standard DP practice has been to maintain two-digit year dates on data files, rather than four-digit year dates. Year 2000 data, when entered, will have to be represented by the two-digit year "00" which will compare and sort as being less than all 20th Century dates previously entered, as well as causing other difficulties. See EXHIBIT A for details.

Charmar Enterprises. Inc.

THE MAGNITUDE OF THE PROBLEM

If this problem is allowed to progress unchecked until the year 2000, the consequences will be disastrous. How could any company operate if its accounts receivable, payroll, and billings systems won't function? If its online modules will not accept the current date? If its production and inventory systems are down? Without a doubt, any company or institution that fails to take corrective action early on will be in severe trouble, because there is no easy way to fix this problem on a short-term basis. A company with even a moderately sized library of personalized software would have to spend a fortune, millions of dollars, to correct this problem on a short-term basis, and it could not be done quickly.

Your company probably has at least 50 production programs for every programmer you employ. If 50% of these programs had to be individually modified (figuring \$300 per modification -- 10 hours at \$30/hr., a very conservative estimate), it would cost your company roughly \$7,500 per programmer for program modifications alone, and additional file update programs and JCL modifications would be necessary for external sorts.

Divide the number of programmers your company employs by 120 and that's roughly how many million dollars you'd have to spend to fix this problem now, and it would take you a minimum of six months to do it, assuming half of your programming staff worked full-time on nothing else. And you are most certainly not alone. This serious problem exists not only in virtually every sizeable American company and civic or governmental institution, its pervasiveness is WORLD-WIDE in scope. Has an unrecognized business problem of greater magnitude ever existed previously?

Charmar Enterprises. Inc.

WHY A SOFTWARE FIX IS HIGHLY UNLIKELY:

The mind-boggling complexity of modifying programs and JCL to process year 2000 data makes this problem prohibitively difficult to handle by software.

To begin with, there presently are no industry standard naming conventions. How could a piece of software tell which fields within a program's data division are date fields? It certainly couldn't assume that a 9(5) COMP-3 field is a date field. What would be the key?

Similarly, one can't distinguish, by looking at JCL and control cards, which sorts have dates as control fields. This sort control card field specification: SORT FIELDS=(1,3,PD,A) may or may not be a date. There is no way to tell.

In order to revise programs to process year 2000 data, a program's logic must be deciphered, working-storage fields physically altered, arrays reworked, and so on.

A great many companies have their own idiosyncrasies, such as using single digit month fields and embedding date representations within product numbers.

Consider that no two programmers code the same way, that coding standards at different installations vary greatly, and that, if nothing is done, there will be MILLIONS upon MILLIONS of different programs in many different languages needing modifications by the year 2000.

The potential input combinations for a piece of software would be virtually infinite.

For these reasons, it is improbable that a software package could be designed to handle this task. Even if it were, it probably would be extremely expensive, and there would still be the expense of reprocessing most of your software.

Fortunately, the matter is now academic, because an effective preventive plan of action -- complete with the necessary "tools" -- is now available at almost no cost.

Companies that employ the Charmar Correction will have no need for such a software package.

Charmar Enterprises. Inc.

WHY NOTHING IS BEING DONE:
(or Don't Blame Your DP Manager)

All data processing managers share a common condition: they are on the firing line and swamped with work. Most installations have years of work backlogged. At any one time, a DP manager has a score of people on his back demanding priority attention for their pet projects.

Therefore data processing is a rough and ready, high-pressure process in which only the most pressing problems are attended to. "If it works, don't mess with it" is an axiom of data processing.

The year 2000 problem is, or should be, the proverbial exception to the rule. At the moment, few if any non-DP managers have even an inkling of the problem. A competent DP manager will be aware that handling year 2000 data might be difficult. But since nobody who matters is screaming about it, to him the problem is highly deferrable.

Because he hasn't been confronted by an urgent need to analyze this problem, he's probably unaware that:

1. A software fix is unlikely; and
2. The best way to attack this problem is through preventive action now.

Because of this, companies, even the high temples of data processing, continue to perpetuate the problem.

Charmar Enterprises. Inc.

THE TIME TO ACT: NOW.

Current program longevity patterns indicate that if you implement the Charmar Correction during 1984, more than 97% of your production programs would be year-2000-compatible by the arrival of the year 2000 without having to have been modified to make them so.

If, however, you waited until the year 1990 to implement the Charmar Correction, only about 75% of your programs would be year-2000-compatible by the year 2000 without needing modification, in other words, 25% of all your programs would have to have been modified, a major waste of time and money.

Depending on the size of your program library, failure to act promptly can cost you thousands or millions of dollars -- more and more the longer you delay.

If your primary programming language is COBOL, you may test this finding by scanning your production program libraries for the literal "DATE-WRITTEN" to compile percentages of existing programs more than 10 or 16 years old.

The relevant question is this: why should your company continue to write programs that are not year-2000compatible when, after a brief initial realignment, it is just as easy to write programs that are?

Obviously the time for the Charmar Correction is now.

Why wait?

Charmar Enterprises. Inc.

THE FOUR-DIGIT YEAR:
PART OF THE SOLUTION.

The "right" way to solve the year 2000 problem would be to require that all new programs contain a four-digit year in every date field, and to devise a way for existing (two-digit year) systems to interface with the new systems.

In a four-digit year system, the year 2000 would compare as being greater than the year 1999, program modifications would be unnecessary, and sorting by date would pose no problem.

Unfortunately, the two-digit year date is too firmly entrenched in existing systems and data files for this to be practical. New programs usually have to refer to existing data files, all of which contain two-digit year dates.

Conversion programs would be necessary for sequential files. Duplicate includes or copy members for file formats would be necessary, and so on.

For these reasons, it is impractical to require that all new programs use four-digit year date fields.

However, there are good reasons for requiring that all new data files created by your company have four-digit year date fields, one being that these fields may eventually accumulate data spanning a century in duration, a built-in limitation of the two-digit year date field.

So much for the general background. Now let's move to the solution and how to apply it at your installation.

8.

Charmar Enterprises. Inc.

THE CHARMAR CORRECTION:
THE BASIC PRINCIPLE:
SOLUTION BY ATTRITION.

Require that, effective immediately, all new programs and JCL must include the necessary logic to process 21st Century dates, and the year 2000 problem will virtually solve itself through attrition by the year 2000.

THE TWO BASIC CORRECTIVE ACTIONS:

1. Require that all new programs written using existing two-digit year date data call the 'YR2000' subroutine for comparisons of dates, as well as date editing, conversions, and elapsed time calculations.
2. Establish the policy that all new data files will contain four-digit year date fields. Use the 'FDYEAR' subroutine for these fields.

Charmar Enterprises. Inc.

RECOMMENDED POLICY
FOR THE YEAR 2000 PROBLEM:

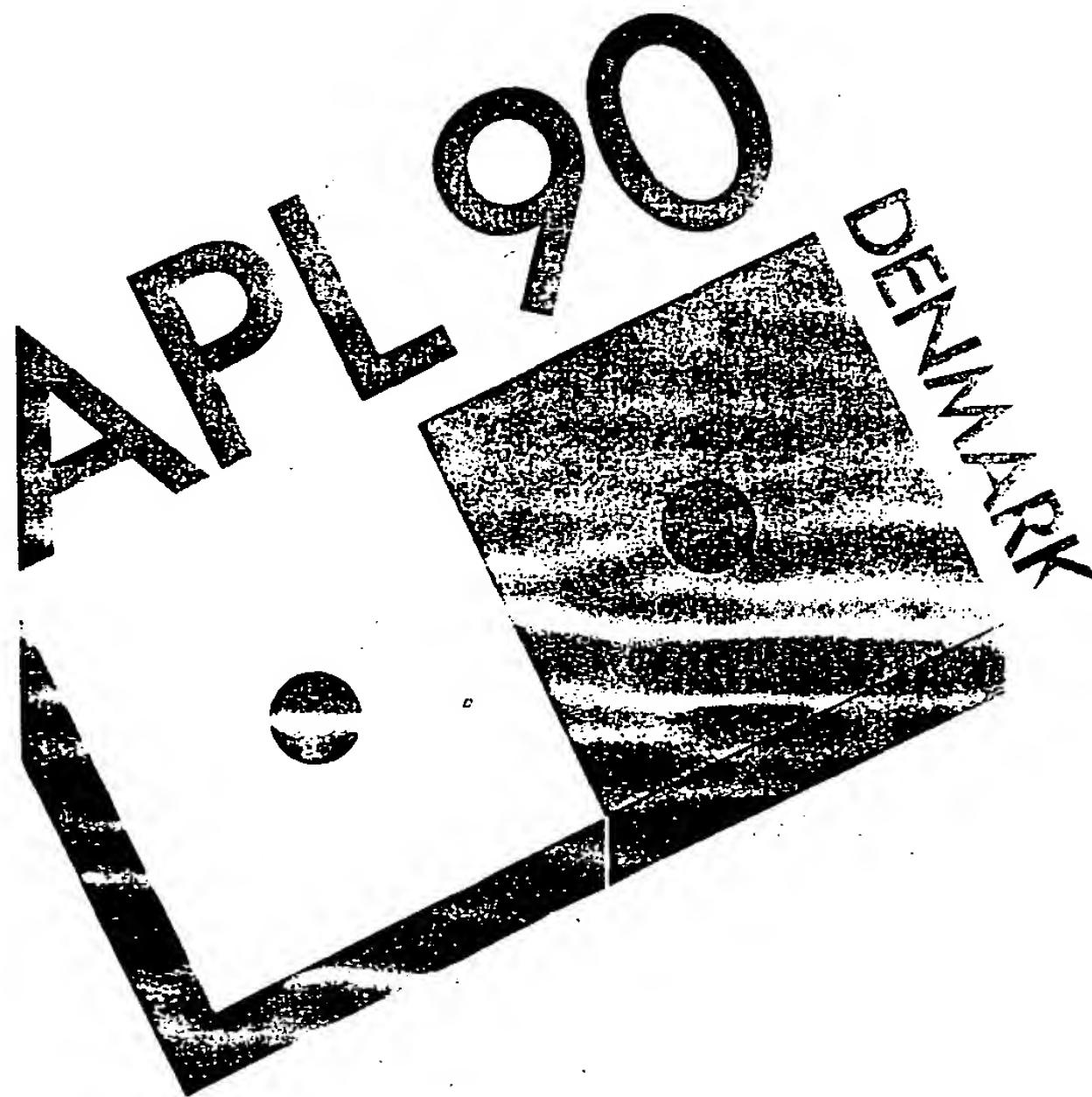
Your company's policy for the year 2000 problem should:

1. Implement the Two-Digit Year Directive (which appears on the following page) utilizing the 'YR2000' subroutine. This procedure addresses the bulk of the problem by correcting current -- and harmful-- standard programming practices based on the two-digit year date field.
2. Require that all new data bases and data files created contain four-digit year date fields. Use the 'FDYEAR' date subroutine for these fields.
3. Forbid the purchase of new programming languages or software packages that are not year-2000-compatible.
4. Identify all non-standard date processing idiosyncrasies existing within your company's DP environment (i.e. systems using product numbers containing date representations, etc.). Determine which of these conditions are (a) not year-2000-compatible and (b) likely to exist by the year 2000. Analyze each for corrective measures and implement them forthwith so that these problems will not continue to be compounded.

Charmar Enterprises. Inc.

THE TWO-DIGIT YEAR DIRECTIVE:
SPECIFIC STEPS FOR REMEDYING
THE YEAR 2000 PROBLEM IN TWO-DIGIT YEAR SYSTEMS.

1. Require that all new programs written contain logic to process year 2000 data. For languages capable of executing a COBOL subroutine (COBOL, PL/I, etc.), this can be accomplished by (a) implementing the 'YR2000' subroutine and (b) distributing the "NEW DATE PROCESSING POLICY" sheet (EXHIBIT B) to all programmers and enforcing compliance to it.
2. A few of your existing date fields may contain dates earlier than the 'YR2000' subroutine default value. Compile a list of these fields and their override values for distribution. (EXHIBIT C).
3. Discontinue writing new programs in other languages until they have been made year-2000-compatible.
4. When sorting records by date, employ a user-exit routine so that revisions will not be needed by the year 2000. (This will only be necessary until the providers of sort software wake up and incorporate this badly needed facility into their packages. See EXHIBIT D.)
5. Consistently mark new year-2000-compatible programs and JCL so they are easily identifiable as such. See EXHIBIT E.
6. Over the years, regularly monitor the volume of "old" programs and JCL remaining. If and when it becomes advisable, you may want to:
 - A. Establish the policy that whenever an "old" program or JCL stream happens to require modification, it must be made JCL year-2000-compatible.
 - B. Actively begin modifying the remaining "old" programs and JCL on a systematic basis so that by the year 2000 all of your software will be year-2000-compatible.
7. Finally, prior to the arrival of the year 2000, simulate test runs of your vital systems with year 2000 input to discover and correct any remaining problems.



FOR THE FUTURE

CONFERENCE PROCEEDINGS

Sponsored by:



Danish Data Association

Co-sponsored by:

 **SIGAPL**



APL QuoteQuad
Volume 20, Number 4
July 1990



**The Association for Computing Machinery
11 West 42nd Street
New York, New York 10036**

Copyright © 1990 by the The Association for Computing Machinery, Inc. Copying without fee is permitted provided that the copies are not made or distributed for direct commercial advantage and credit to the source is given. Abstracting with credit is permitted. For other copying of articles that carry a code at the bottom of the first page, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 27 Congress Street, Salem MA, USA 01970. For permission to republish, write to Director of Publications, The Association for Computing Machinery. To copy otherwise, or republish, requires a fee and/or specific permission.

ISBN 080-89791-371-X
ISSN 0-163-6006

Additional copies of these proceedings may be ordered prepaid from

ACM Order Department
Waverly Press
P.O. Box 64145
Baltimore, MD
USA 21264

Prices:

Members	\$24.00
Others	\$32.00

When ordering, specify the ACM Order number: 554900.

Table of Contents

Jan Ahlqvist	1
SRS Service Report System	
Manuel Alfonseca	2
Neural Networks in APL	
Manuel Alfonseca	7
Object Oriented Programming, Tutorial	
Jake Ansell	9
ASL - An APL Statistical Library	
Heikki Apiola, Pirkka Peltola	10
Integrating APL with symbol manipulation, numerical software and graphics	
Guy Barker, Douglas J. Keenan, Herman van Loon	18
Conscientious Programming Using PMA	
J. Philip Benkard	27
Nonce Functions	
Robert Bernecky, Charles Brenner, Stephen B. Jaffe, George P. Moeckel	40
ACORN: APL to C on Real Numbers	
P. Bottoni, P. Mussio, M. Protti	50
Definition of Image Interpretation strategies in APL	
Renato Capra	61
Preliminary Mesh Checking for Structural Analysis	
Edward Cherlin	71
APL Trivia	
Wai-Mee Ching	76
Automatic Parallelization of APL-style Programs	
Janice H. Cook, Leo H. Groner	81
Analytic Response Time Model for Distributed Systems	
Stephen W. Dunwell	102
APL as the Foundation for a Universal Computer Language	
Don Erickson	103
Technical Support Program for APL Related Questions	
Frank Evans, Jan Jantzen	104
A Structured Approach to Analysis and Design of Complex Systems	
Ian Feldberg	105
Angiogram Analysis in APL: A Case Study	
William G. Foote, Pamela J. Baron	106
APL2 Analysis and Design of Mortgage Backed Securities	
Garth H. Foster, Abdelatif Elgouri, Franklin Liu	114
Editing and Debugging with Windows and a Mouse	
Ralph L. Fox	121
Color APL Beautiful!	

Erik S. Friis, Stanley Jordan	130
Musical Syntactic and Semantic Structures in APL2	
Andreas Geyer-Schulz, Johann Mitlöhner, Alfred Taudes	140
An APL-Simulator of Non-Von Neumann Computer Architectures	
Jean-Jacques Girardot	149
The A+ Programming Language, A Different APL	
Jean-Jacques Girardot	161
Arrays and References	
Alan G. Hawkes	173
Markov Processes in APL	
Ferdinand Hendriks, Wai-Mee Ching	186
Sparse Matrix Technology Tools in APL	
Roger K. W. Hui, Kenneth E. Iverson, E. E. McDonnell, Arthur T. Whitney	192
APL\?	
Torben Iversen	201
APL for economic and management control in KTAS	
Eeva-Liisa Kaski	206
End-User KESSU	
Andrew V. Kondrashev	214
The Family of Soviet APL Systems	
Morten Kromberg, Martin Gfeller	217
An Application Development Platform	
Gérard A. Langlet	228
The Travelling Salesman Problem, revisited with APL	
Timo Laurmaa	233
Desktop Publishing on the Mainframe: Integrating APL2 and Ventura Publisher	
Edward Y. H. Lin, Dennis L. Bricker	239
Implementing the Recursive APL Code for Dynamic Programming	
Jim Lucas	251
Programming Ecology or APL and the World at Large	
Sven Gunnar Lönn	260
DEMOS - a PC-system for Population Projections for small areas	
Ole M. Meyer	263
An Insurance Simulation Model	
Alexey L. Miroshnikov	271
Algorithm Alterable Models and APL	
Trenchard More, Jr.	274
An Array-theoretic Look Beyond APL2 and Nial	
Thomas M. Olsen	279
Multi-Axis NC Postprocessor for Machining Centers	
Panagiotis Pantziarka	284
Object Oriented Database using Frames in Second Generation APL	

Raymond P. Polivka	288
Reading to Write	
Thomas J. Pritchard	298
IBM System/370 Channel Programming Using APL	
Steven I. Promisel, James V. Merrill	304
Managing a Diamond Jewelry Manufacturing Business using APL	
Ursula Recker, Michael Rys	312
A Preferable Look - APL in Window-based Environments	
Jack G. Rudd, James A. Brown	322
Toward a Common Prototyping Language	
William A. Rutiser	331
Object-Oriented Programming of a Window System Graphical User Interfaces	
Jürgen Sauermann	342
A Parallel APL Machine	
Charles A. Schuiz	348
Writing Applications for Uniform Operation on a Mainframe or PC: A Metric Conversion Program	
D. Smellie, F. Evans	362
Structured Expert System Design	
Howard J. Smith, Jr.	363
Some Uses of Truncated Boolean Vectors in Analysis	
Howard J. Smith, Jr.	364
What's Ahead for 2000 A.D.?	
Donald Soule	365
Stability in a Sea of Change	
Thomas L. Springall, Gustav Tollet	369
A Shading Approach to Non-Convex Clipping	
Timo Teileri, Toivo Olkkola	373
LYYTI - Integrated Design and Control System	
Norman Thomson	378
APLELEGANCE - The Art of Staying Within One's Depth	
J. W. B. Vermeulen, E. R. K. Spoor	391
FRESH, an Expert System Design Tool on APL2	
David M. Weintraub	404
APL2OS: Design Considerations for a Nested Array File System	
James G. Wheeler	412
Design and Implementation of the Interface to Compiled Languages in APL*PLUS II	
Jonny Österman	418
Very High Quality User Interfaces and Fast Data Filing using a PC	
Author index	429
Index	430
Sponsors	435

What's Ahead for 2000 A.D.?

Howard J. Smith, Jr.
IBM Corporation
1501 California Avenue
Palo Alto, CA 94304

On January 1, 2000 many applications will stop working properly. Most utilities and applications express the year as the two low order digits of the full representation of the year so that dates in the new century will appear to occur before dates in the old. When this phenomenon occurred in the past humans adapted quickly, and no great dislocation occurred.

Unfortunately, the computer is not as adaptable or reasonable as the human.

This paper presents one possible solution to the problem, at least in the APL environment. This solution involves replacing date fields in files and records with a pseudo-Julian day number (PJD). This number permits the analyst to use date information arithmetically to determine the day of the week, the period between dates, create lists of dates, etc. The PJD is easily converted to or from human-usable

canonical date forms. Finally, the technique defers difficulties due to changes of year, century, millennium, etc. indefinitely.

In the later sections of the paper some thoughts on possible extensions to the leap-year algorithm are discussed. The new terms would improve precision without affecting the past or present. Finally, sample functions which support and utilize this "perpetual" calendar are given.

It must be emphasized that this paper represents a solution only within the APL environment. The concept is applicable in any environment, language or system but only if it or a similar solution is adopted before December 31, 1999. If we users of APL recognize and fix the problem in our environment while the rest of the computing world ignores it, the problem will still affect everyone, including us.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

• 1990 ACM 089791-371-x/90/0008/0364...\$1.50

#13
47
InformationWeek

February 26, 1990, Issue: 259

Section: TE

PREPARING FOR 2000 -- On the first day of the next decade, millions of Cobol programs may be wrong
John Xenakis

Hidden within the billions of lines of code that make up software programs across the nation is a time bomb. On the first day of the next decade, the first of the next millennium, millions of computer programs written in Cobol, PL/1, and other languages, will start giving out the wrong answers.

The problem is the way in which standard programs have represented dates. The only standard way the current date is identified in a Cobol program returns the date in the six-digit form YYMMDD, with two digits for the year. Feb. 26, 1990, for example, is represented as 900226, and Jan. 1, 1991, is represented as 910101. The former is smaller than the latter, so the computer assumes that the first date is earlier than the second one. And almost all Cobol programs depend on that fact to determine the answers to questions related to such things as inventory expiration and interest computations.

But on Jan. 1, 2000--that is, 000101--those comparisons will fall apart. Programs which depend on such computations will get the wrong answers. Even standard sort utilities, which arrange records of a file in order by date, will fail. In order to work correctly, these programs and files will have to be modified to store a four-digit year, such as 19900226 or 20000101, in the format YYYYMMDD.

"About 75% of all the Cobol programs out there are going to have to be modified," says William Strapko, an analyst with market researcher International Data Corp. in Framingham, Mass. "The biggest problem is to convince management that they're going to have to dedicate a fair amount of resources in order to resolve this problem. It won't be easy."

"Can you recompile old programs and have them run correctly?" asks Carl Cargill, a standards consultant at Digital Equipment Corp. in Maynard, Mass. "That will be affected by how well the programs were written. The IS managers who hired cheap entry-level programmers got the job done cheaply, but now they're going to have to pay for it."

Of course, many programs were written carefully to start with. "In my last life, I used to work on an international banking system," says Marc Sokol, executive VP at Chicago-based Realia Inc., a vendor of Cobol compilers. "We had to keep track of maturities that expired in 20 years, so we were very aware of the problem."

Bill Schoen, an independent consultant in Clarkston, Mich., points out, however, that such prescience is rare. "I've worked as a consultant to Chrysler, Ford, and EDS Electronic Data Systems Corp.!", he says, "and I know they've done very little so far to handle this problem. I can guarantee that even many systems being written today are going to fail in 2000."

DEC's Cargill says computer and compiler vendors are going to be under pressure to help their customers. "The user will say, 'I have a lot of money invested in your equipment, and I'm going to invest a lot more. What are you going to do to fix this?' The problem is that the user will have to convert his own programs."

To aid conversions, the ANSI Cobol standard has recently been modified to accommodate a new intrinsic function named Current-Date, which gives the current date with a four-digit year. Vendors are required to implement this function within two years if they wish to remain compliant.

Some IS managers are hoping for a program that will convert old Cobol programs to use the new intrinsic function automatically, but Schoen says a lot of manual work is still needed. "Even if new technology caught 98% of all the errors, the other 2% will be a disaster."

In fact, there is a precedent for this situation. In 1979, the U.S. Postal Service went from a five-digit zip code to a nine-digit zip code. "It was a horror show," says Cargill.

The difference, of course, is that you don't have to change your zip code software. The year 2000 won't offer that choice.